TREBALL FINAL DE GRAU

TÍTOL DEL TFG: E-bet: Joc distribuït i tokenitzat amb *Blockchain*

TITULACIÓ: Grau en Enginyeria Telemàtica

AUTOR: Aleix Cánovas Esteban

DIRECTOR: Juan Hernández Serrano

DATA: 6 de setembre del 2019

Títol: E-bet: Joc distribuït i tokenitzat amb *Blockchain*

Autor: Aleix Cánovas Esteban

Director: Juan Hernández Serrano

Data: 6 de setembre de 2019

Resum

En els últims anys, l'ús de la tecnologia *Blockchain* s'ha expandit en gran mesura. Aquest fet s'ha produït gràcies a la distribució, immutabilitat, transparència i seguretat que aporten aquest tipus de xarxes.

L'aplicació més significativa de les xarxes *Blockchain* són les criptomonedes. Però, amb la introducció de la xarxa *Ethereum* i la creació dels *Smarts Contracts*, els desenvolupadors poden realitzar aplicacions en tants àmbits diferents com idees en tinguin.

E-bet és una aplicació que es beneficia dels avantatges que aporta la xarxa *Ethereum* per crear un joc d'apostes distribuït i tokenitzat. *E-bet* és una plataforma on es poden interconnectar diferents jugadors *online*, mitjançant un joc d'apostes un contra un sobre partides del videojoc *League of Legends*. Per diferenciar-se, *E-bet* dóna l'opció, no només d'apostar en partides professionals, sinó que permet realitzar una aposta en qualsevol partida que estigui jugant qualsevol jugador.

Per realitzar una aposta és necessari obtenir l'actiu que es vol apostar. Per aquest motiu, s'ha creat el token *ERC-20* anomenat *E-bet*. Amb la creació del token *E-bet*, i els avantatges introduïts anteriorment, es poden gestionar tots els balanços de cada compte i administrar totes les transferències de *tokens* entre adreces sense la necessitat d'una entitat central.

En aquest projecte, s'han realitzat tots els passos per poder crear l'aplicació descrita. S'han desenvolupat els *Smarts Contracts* que permeten gestionar tots els estats de les apostes així com els balanços de cada compte. També, s'ha desenvolupat el *Back-End* on es gestiona tota la informació de les partides, apostes i usuaris que es volen guardar a la base de dades, o que es volen ensenyar a l'aplicació. Finalment, s'ha desenvolupat una aplicació que permet crear i acceptar apostes des del dispositiu de l'usuari i modificar directament la *Blockchain* utilitzant una *Wallet* creada.

Títol: E-bet: Joc distribuït i tokenitzat amb *Blockchain*

Autor: Aleix Cánovas Esteban

Director: Juan Hernández Serrano

Data: 6 de setembre de 2019

Overview

In recent years, the use of Blockchain technology has expanded greatly. This fact has occurred thanks to the distribution, immutability, transparency and security that this type of network provides.

The most significant application of the Blockchain networks are cryptocurrencies. But, with the introduction of the Ethereum network and the creation of Smarts Contracts, developers can make applications in as many different areas as ideas they have.

E-bet is an application that benefits from the advantages of the Ethereum network to create a distributed and tokenized betting game. E-bet is a platform where you can interconnect different players online, using a one vs one betting game of the League of Legends videogame. In another point of view, E-bet gives the option not only to bet on professional games but also to make a bet on any game that any player is playing.

To make a bet, it's necessary to obtain the asset that you want to bet. For this reason, the token ERC-20 called E-bet was created. With the creation of the E-bet token and the advantages introduced above, all balances of each account can be managed and all transfers of tokens between addresses without the need of a central entity can be done.

In this project, all the steps have been taken to create the described application. The Smarts Contracts have been developed to allow to manage all the states of the bets as well as the balances of each account. Also, the Back-End was developed to manage all the information on the games, bets and users that are wanted to be stored in the database, or wanted to be shown in the application. Finally, an application has been developed in a way that allows users to create and accept bets from the user's device and directly modify the Blockchain using a created Wallet.

ÍNDEX

INTRODUCCIÓ

La tecnologia *Blockchain* es troba en un continu creixement i és una de les principals tecnologies per la qual les empreses inverteixen els seus diners. Aquesta tecnologia permet a les empreses descentralitzar processos i obtenir una traçabilitat perfecta de tots els seus actius físics i digitals. Però, *Blockchain* no només s'utilitza en l'àmbit econòmic. Amb la introducció dels *Smarts Contracts d'Ethereum* es poden desenvolupar qualsevol mena d'aplicació i aplicar-ho a qualsevol àmbit.

La diversitat en vers a crear qualsevol mena d'aplicació té un cert risc. És necessari realitzar un estudi sobre la viabilitat i la necessitat d'utilitzar Blockchain, ja que, per molts avantatges que aquestes xarxes disposin, també tenen importants problemes.

Una xarxa *Blockchain* és descentralitzada, pública, immutable i segura amb l'ús de criptografia asimètrica. No depenen d'un punt central, tots els participants poden veure informació (no informació personal) i les transaccions no es poden ni modificar ni esborrar. Aquests fets els fa ser molt més segures i justes que qualsevol altre tipus de xarxa.

Però, com s'ha comentat amb anterioritat, tots aquests avantatges poden tenir el seu punt en contra. Com és descentralitzada, cada usuari ha de tenir la *Blockchain* utilitzada sincronitzada en tot moment. També, encara que no es filtri informació personal, es pot tenir una traçabilitat de les transaccions que cada usuari realitza. A més, al ser immutable, si s'ha produït o generat un error en el codi, aquest no es pot eliminar i pot ser explotat per qualsevol usuari.

Paral·lelament a l'àmbit de la tecnologia *Blockchain*, i en un sector totalment diferent, els *e-sports* estan en augment exponencial. Amb la introducció de diverses i noves plataformes de jocs i amb la creació de lligues professionals, els *e-sports* han elevat considerablement el nombre de jugadors i seguidors.

Tenint en compte els arguments tecnològics introduïts anteriorment i el sector en creixement comentat, la idea d'aquest projecte és crear una plataforma *online* per connectar qualsevol jugador del món dels *e-sports*. Aquesta plataforma consisteix en un joc on no només s'incentiva en participar o en visualitzar partides, sinó que es dóna un estímul lúdic als usuaris d'aquesta aplicació.

La funcionalitat principal de l'aplicació és crear unes apostes un contra un entre dos usuaris on es juguen aconseguir una recompensa en forma de *tokens*. Aquests *tokens* estan basats en l'estàndard ERC-20, que consisteix en una sèrie de normes que defineixen la gestió de balanços de cada adreça i l'administració de les transferències dels *tokens*.

Resumint, aquest projecte consisteix a crear una plataforma on poder interconnectar diferents jugadors *online* mitjançant un joc d'apostes descentralitzat i criptogràficament segur.

Situació actual / problemàtica

Avui en dia, existeixen un gran nombre de plataformes per poder realitzar apostes sobre qualsevol esdeveniment competitiu. En un passat no gaire llunyà, només es podia realitzar apostes en competicions estrictament esportives. Però, des de ja fa uns anys, diferents empreses permeten als seus clients la possibilitat de realitzar apostes sobre altres esdeveniments, com per exemple, els *e-sports*.

La intenció de *E-bet* és continuar per aquest camí i donar la possibilitat de tenir una plataforma exclusivament per realitzar apostes sobre els *e-sports*.

A més, *E-bet* es vol desmarcar de la resta de la competència donant la possibilitat de passar al següent nivell. No només ofereix la possibilitat de realitzar apostes en les partides dels professionals, sinó que a més, ofereix la possibilitat d'apostar en qualsevol partida en línia jugada per qualsevol jugador.

Objectius

L'objectiu principal d'aquest projecte és la creació d'una plataforma capaç d'interconnectar i impulsar als usuaris a participar en el joc d'apostes relacionades amb els *e-esports*.

Com s'ha comentat amb anterioritat, l'aplicació *E-bet* permet realitzar apostes no només en partides professionals, sinó que permet realitzar una aposta de qualsevol partida. Aquest fet és la proposta de valor i la diferència més important respecte a una casa d'apostes convencional.

Un dels objectius més importants d'*E-bet* és permetre als usuaris tenir un *token* intercanviable per *Ethers* amb el qual poder realitzar aquestes apostes de manera segura. Amb l'aplicació i el desenvolupament dels *Smart Contracts* que es volen crear, i tota la infraestructura que ens aporta la xarxa *Ethereum*, es vol i es pot obtenir aquesta seguretat.

A part, un dels objectius de l'aplicació *E-bet* és poder gestionar tots els usuaris i totes les apostes que es realitzen, juntament amb tota la quantitat d'informació que s'obtindrà de les partides que es buscaran.

Finalment, un altre dels objectius amb més pes és obtenir benefici econòmic. Totes les aplicacions desenvolupades busquen diferents maneres d'obtenir benefici. *E-bet* té com a objectiu trobar un benefici econòmic sense aprofitar-se dels usuaris, ni amb la intenció de proporcionar-los una mala experiència d'usuari.

Estructura del document

El document està dividit en els capítols que es descriuen a continuació:

- CAPÍTOL 1: Conceptes Previs. En aquest capítol s'expliquen els conceptes bàsics per poder entendre l'explicació sobre *Blockchain* i el projecte.
- CAPÍTOL 2: *Blockchain – Ethereum*. En aquest capítol s'explica el significat de *Blockchain* i les seves característiques, així com els aspectes més importants *d'Ethereum* i de les aplicacions descentralitzades.
- CAPÍTOL 3: Arquitectura de desenvolupament. En aquest capítol s'explica l'estructura del projecte i totes les tecnologies utilitzades per desenvolupar el projecte.
- CAPÍTOL 4: *E-bet*. En aquest capítol es descriu l'aplicació i les seves funcionalitats principals. A més, s'expliquen els mecanismes d'incentius i es comenta la viabilitat econòmica i legal de l'aplicació desenvolupada.
- CAPÍTOL 5: *Smart Contracts*. En aquest capítol s'expliquen en detall tots els *Smart Contracts* desenvolupats: El contracte principal *Bet.sol*, les funcions del token *ERC20.sol* i la llibreria *SafeMath.sol*.
- CAPÍTOL 6: *Back-End*. En aquest capítol s'explica els models i la connexió amb la base de dades, *l'API* creada, la connexió i la utilització de *l'API* de *Riot Games*, el *Cron*, els *Sockets* i les mesures de Seguretat implementades.
- CAPÍTOL 7: *Front-End*. En aquest capítol es descriu la navegació de l'aplicació, s'explica totes les seves pàgines i la *wallet* desenvolupada.
- CAPÍTOL 8: Conclusions i línies futures. En aquest capítol s'exposen els passos futurs que s'han de realitzar per millorar i publicar el projecte. A més, s'expliquen les conclusions extretes en realitzar el desenvolupament del projecte.

Addicionalment, s'ha realitzat uns annexos per complementar el projecte:

- Annex A: Guia d'usuari
- Annex B: Mecanismes de consens
- Annex C: Normativa *Riot Games*
- Annex D: Descripció de les tecnologies utilitzades
- Annex E: Fluxos – Funcionalitats futures
- Annex F: Models de dades a guardar a MongoDB
- Annex G: Funcions Socket – Xat

A més a més, es pot consultar el codi obert en els següents repositoris de *Github*:

- *Back-End*: https://github.com/Aleix11/server-tfg
- *Front-End*: https://github.com/Aleix11/client-tfg-ionic

CAPÍTOL 1. CONCEPTES PREVIS

En aquest capítol s'expliquen els conceptes previs que es necessiten saber abans de definir què és una xarxa *Blockchain* i posteriorment poder entendre millor l'aplicació desenvolupada. S'expliquen les definicions de xarxa, computació i base de dades distribuïda a més de la definició de *hash* i de criptografia de clau pública.

1.1. Xarxa distribuïda

Una xarxa distribuïda és una topologia de xarxa caracteritzada per l'absència d'un centre individual o col·lectiu. Els nodes s'uneixen entre ells de forma que cap d'ells té el poder de filtrar les dades que es transmeten.

Aquest tipus de xarxes són robustes respecte a la caiguda de nodes. Això significa que si algun node cau, té algun problema, o vol sortir de la xarxa, no genera la desconnexió dels d'altres.

1.2. Computació distribuïda

La computació distribuïda és un model informàtic que permet fer grans càlculs i resoldre grans problemes de computació utilitzant un gran nombre d'ordinadors. Aquest sistema es fonamenta en dividir o distribuir el treball a realitzar en petites feines, per tal d'estalviar costos i temps.

Es tracta d'un model de computació coordinada on interactuen entre ells una col·lecció d'ordinadors que poden o no estar ubicats en llocs diferents.

Característiques:

- Seguretat interna en el sistema distribuït.
- S'executa en múltiples ordinadors.
- Existeixen diverses còpies del mateix sistema operatiu que proveeixen els mateixos serveis.
- Els ordinadors no han de ser homogenis.
- Compatibilitat entre els dispositius connectats.
- Transparència.
- Interacció entre els equips.

1.3. Base de dades distribuïda

Es tracta d'un conjunt de diverses bases de dades interrelacionades i situades en diferents nodes d'una xarxa d'ordinadors. Els múltiples ordinadors que s'utilitzen en aquest tipus de bases de dades s'anomenen nodes. Aquestes

bases de dades tenen la capacitat de realitzar processaments autònoms, i realitzar operacions de forma local o de manera distribuïda.

Avantatges:

- Localització transparent de les dades: Totes les instruccions no depenen ni del lloc on s'executen, ni del lloc on se situen.
- Transparència en les dades i en la fragmentació: Es pot accedir als objectes d'aquestes bases de dades a partir de qualsevol camp. A més, la fragmentació d'aquestes dades és transparent, de forma que es pot seguir un rastre de les dades segons la localització d'aquestes bases de dades.
- Millora en la fiabilitat i disponibilitat: Les dades i els softwares estan distribuïts en diferents ordinadors. Si un ordinador falla, els altres poden seguir funcionant sense cap problema. Però, les dades d'aquest ordinador, són inaccessibles. Per solucionar aquest problema, es realitzen rèpliques automàtiques en diferents ordinadors.
- Millora del rendiment: S'intenta distribuir les dades en els ordinadors on s'utilitzen o en els més propers.
- Expansió més senzilla: Les funcions d'afegir dades, d'afegir processadors o d'augmentar la base de dades són tasques més senzilles de realitzar.

1.4. *Hash*

Les funcions *hash*, o funcions resum, són algoritmes que converteixen una entrada qualsevol de longitud variable en una sortida de longitud fixa, normalment menor a l'entrada. El resultat obtingut seria un resum de l'objecte inicial.

Les funcions de *hash* han de complir una sèrie de requisits, per a poder ser considerades com a tal:

- Unidireccionalitat: donada una *x*, ha de ser fàcil calcular *h(x)*, però contràriament, ha de ser difícil trobar *x* a partir de *h(x)*.
- Resistència a col·lisions: és a dir, que sigui molt difícil trobar una parella *x* i *y*, amb $x \neq y$ que tinguin la mateixa sortida.
- Determinisme: això vol dir que una mateixa entrada sempre donarà com a resultat la mateixa sortida.
- Difusió: el resum de *h(x)* ha de ser una funció de tots els bits de l'entrada.
- Facilitat de càlcul: calcular *h(x)* ha de tenir baix cost.

Les funcions de *hash* s'utilitzen principalment per a temes d'integritat i autenticació de la informació. En temes d'integritat, serveixen per comprovar si un missatge ha sigut modificat o no. Quan es rep el missatge, es calcula el resum i es compara amb el *hash* rebut, i si són diferents, vol dir que ha sigut modificat.

En termes d'autenticació, es fa servir junt amb la firma digital per a verificar l'origen dels missatges. L'emissor adjunta un *hash* del missatge signat amb la clau privada, i quan l'emissor el rep, calcula el resum del missatge i desxifra el hash rebut, per comparar els dos, i verificar que ha sigut enviat per l'emissor.

1.4. Criptografia de clau pública

La criptografia de clau pública és un mètode criptogràfic que es basa en l'ús d'un parell de claus per a l'intercanvi de missatges. Cada individu disposa de dos claus diferents, una anomenada clau privada, que només coneix ell, i una pública, a la qual tindran accés altres usuaris que vulguin interaccionar amb aquest. Els missatges xifrats amb una de les claus només es poden desxifrar amb l'altra, i viceversa.

La criptografia de clau pública s'utilitza principalment en dos casos:

- Encriptació de missatges: si algú vol enviar un missatge de manera segura a una altra persona, utilitza la clau pública d'aquesta per xifrar els missatges, de manera que només els pot desxifrar el destinatari amb la seva clau privada.
- Firma digital: l'emissor utilitza la seva clau privada per signar el missatge, i quan el receptor el rep, pot verificar amb la clau pública de l'emissor que el missatge ha sigut enviat per ell, i no ha sigut modificat pel camí.

CAPÍTOL 2. *BLOCKCHAIN – ETHEREUM*

En aquest capítol s'explica tota la informació necessària per poder entendre què és una xarxa *Blockchain*. Inicialment, es defineix el concepte i s'exposen les característiques principals i, a continuació, s'explica informació essencial sobre les transaccions, els blocs, els *Smarts Contracts* i les aplicacions.

Seguidament, s'explica la definició de la xarxa *Ethereum* incloent-hi aspectes com el *World State*, les transaccions, els blocs, les aplicacions i diferents *tokens*. A més, s'entra en detall amb el *token* utilitzat en aquest projecte, el *token ERC-20*.

Finalment, s'explica què és una aplicació descentralitzada (*DApp*), els diferents tipus i les diferències entre una *App* i una *Dapp*.

2.1. *Blockchain*

2.1.1. Definició

Una *Blockchain* és una combinació dels conceptes explicats en el punt anterior. Es tracta d'una llista de registres distribuïda formada per una sèrie de cadenes de blocs amb la intenció d'evitar la modificació de qualsevol dada afegida. En aquestes cadenes de blocs s'utilitza una espècie de "segell" i s'afegeix informació relativa als blocs anteriors de la cadena mitjançant criptografia. Aquesta tecnologia, a més, manté l'anonimat dels usuaris i fa un registre de totes les dades per poder mantenir la seva integritat i confiança.

Qualsevol *Blockchain* (sigui privada o pública) té un caràcter públic. Aquest fet, permet que qualsevol participant de la xarxa pugui veure les dades que s'hi afegeixen. Però, aquest caràcter transparent de la xarxa no significa que no hi hagi certs nivells de privacitat. Una *Blockchain* proporciona informació pública del seu estat general i de la informació continguda sense filtrar cap mena d'informació personal de cada usuari. Per mantenir els diferents nivells d'anonimat, les xarxes *Blockchain* utilitzen criptografia de clau asimètrica (criptografia de clau pública). La clau pública és la direcció de cada usuari de la *Blockchain*, i la clau privada es tracta del paràmetre que dóna accés als actius digitals de cada usuari i permet firmar les dades que es volen pujar a aquestes xarxes.

2.1.2. Característiques principals

- La informació emmagatzemada es distribueix en molts ordinadors, anomenats nodes. Es guarden rèpliques en temps real entre tots els nodes, que contenen el mateix valor, i la mateixa importància dins de la xarxa.

- La transmissió de dades s'aconsegueix via *peer-to-peer.* Aquest tipus de transmissió és una xarxa d'ordinadors on no existeixen ni clients ni servidors fixos, sinó que es tracten d'una sèrie de nodes que es comporten de la mateixa manera.
- La verificació de dades s'aconsegueix mitjançant un procés de consens entre els nodes. Existeixen diferents tipus de verificacions de dades (explicades en l'apartat 2.1.5. Blocs).
- Les dades són immutables, ja que no es permet modificar ni esborrar cap dada continguda a la *Blockchain*
- Cada usuari d'una xarxa *Blockchain* té associada una adreça. Aquesta adreça és l'identificador del compte de l'usuari i permet l'accés a una cartera *Blockchain* (*wallet*). Una *wallet* és un punt de referència vinculat a una aplicació on es pot gestionar les dades que afegeix/transmet cada usuari. Una *wallet* per la *Blockchain* seria com un navegador per *Internet*.
- Cada dada afegida a la *Blockchain* està signada criptogràficament pel seu emissor.
- La xarxa és transparent, ja que es tracta d'un registre on es guarden les dades i només es poden consultar (no modificar). Per tant, es garanteix la integritat de la informació.

2.1.3. Tipus de *Blockchain*

- *Blockchains* públiques: Aquest tipus de *Blockchain* no tenen absolutament cap restricció d'accés. Qualsevol persona amb connexió a Internet i un compte *Blockchain* pot carregar dades o convertir-se en verificador.

  Aquestes xarxes poden oferir una compensació econòmica pels verificadors (per exemple: compensació per la resolució de problemes computacionals).

  Exemples: *Bitcoin*, *Ethereum*, etc...

- *Blockchains* privades: En aquest tipus de *Blockchain* es requereix un permís per unir-se. No es pot entrar si un administrador no et convida.

  Aquestes *Blockchains* es poden considerar com un registre distribuït de dades per a empreses, ja que pot incorporar processos de comptabilitat i manteniment de registres.

  Per exemple: *Hyperledger*, *Quorum*, *Kaleido*.

- *Blockchains* de consorci: Aquest tipus de *Blockchain* són xarxes semi-descentralitzades o semiprivades. També, i com les *Blockchains* privades, es necessita estar autoritzat per poder entrar-hi. El control d'aquestes no està centrat en un administrador, sinó que un nombre de nodes/empreses porten el control d'aquesta xarxa.

  Les empreses d'una *Blockchain* de consorci poden restringir els drets de verificació dels usuaris tal com considerin oportuns, i només permeten,

en general, que un conjunt limitat de nodes executin el protocol de consens o validació de dades.

Per exemple: *Alastria* (Consorci espanyol amb nodes regular i verificadors).

2.1.4. Transaccions

Les dades que s'afegeixen a qualsevol *Blockchain*, s'anomenen transaccions. Però, el seu objectiu pot no ser el mateix que una transacció financera comuna. Una transacció és una secció de dades amb firma digital que es transmeten a la xarxa i es guarden en els blocs. Cada transacció està formada, com a mínim, pels següents camps:

- *Hash*: És un valor identificatiu d'una transacció.
- *Value*: És el valor monetari que es transmet en una transacció.
- *Receiver*: És el *hash* identificatiu del receptor d'una transacció.
- *Sender*: És el *hash* identificatiu de l'emissor d'una transacció.
- *BlockHash*: És el *hash* identificatiu del bloc on es guarda una transacció.
- *BlockNumber*: És el número que té un bloc dins de la *Blockchain*.
- *TransactionIndex*: És la posició que ocupa la transacció dintre del bloc.

2.1.4.1. Recorregut d'una transacció:

Qualsevol transacció que es vol fer en una xarxa *Blockchain*, ha de seguir un camí per poder-se completar.

- S'envia la transacció a la *Blockchain*.
- La transacció s'emet entre els diferents nodes en aquesta xarxa *peer-to-peer.*
- Diferents nodes de la *Blockchain* verifiquen la transacció seguint un algoritme de verificació.
- Aquesta transacció verificada s'introdueix en nou bloc, que és introduït a la cadena de blocs.

2.1.5. Blocs

Un bloc és un paquet de dades. Conforma l'estructura bàsica on s'emmagatzemen una o diverses transaccions. Els blocs d'una xarxa *Blockchain* s'organitzen en una seqüència lineal al llarg del temps on cada bloc apunta a l'anterior. Aquesta seqüència s'anomena cadena.

Les dades de dins d'un bloc estan representades mitjançant *hashs* i cada bloc nou que s'afegeix a la cadena, hereta part dels *hashs* de l'anterior. Això s'aconsegueix construint *tries* formats pels *hashs* de tots els blocs anteriors.

Un *trie* és una estructura en forma d'arbre on les dades es guarden als nodes fulla i, mitjançant *hashs*, s'obtenen nous nodes a partir de grups de *n* nodes. Aquest procés s'aplica iterativament fins a obtenir un únic node, anomenat node arrel/gènesis, amb un sol *hash*.

Gràcies a aquesta estructura de dades conformada per *tries* construïts recursivament pel *trie* anterior, podem verificar amb certesa la integritat i la validesa de tota la cadena de blocs. Aquesta és una de les propietats més importants de les xarxes *Blockchain* i és degut precisament a la seva estructura i construcció.

2.1.5.1. Mecanismes de consens:

A continuació, ens centrarem en els diferents tipus de confirmació/validació de blocs en les xarxes *Blockchain*. Aquests tipus de validacions, s'anomenen mecanismes de consens.

Un mecanisme de consens és una forma de garantir un acord mutu. Dins de les *Blockchain*, els mecanismes de consens asseguren que cada node de la xarxa té una còpia de les dades. Però, no existeix només un tipus de mecanisme de consens.

El mecanisme de consens més famós es tracta el de *Proof of Wor*k (*PoW*, l'utilitzat per *Bitcoin*, *Ethereum*, etc...). A més, existeixen altres mecanismes de consens com: *Proof of Elapsed Time* (*PoET*), *Proof of Authority* (*PoA*), *Proof of Stack* (*PoS*), *Proof of Capacity* (*PoC*), *Proof of Activity*, *Proof of Burn*, i molts d'altres.

En l'*Annex B: Mecanismes de consens,* es troben descrits els mecanismes de consens més famosos i més utilitzats.

2.1.6. *Smart Contracts*

Per entendre què és un S*mart Contract* (contracte intel·ligent) primer recordarem què és un contracte legal. Un contracte legal és un acord entre dues parts, com a mínim, on es descriu aquest acord, com es pot accionar, produir i operar, i que succeeix en cas d'incomplir-lo. Normalment els contractes legals estan escrits en llenguatge jurídic i/o tècnic i malgrat vulguin ser el més objectiu possible sempre poden estar subjectes a interpretació de qui els executa.

A diferència d'un contracte legal, un S*mart Contract* és un algorisme que s'executa de manera autònoma i automàtica quan es compleixen unes condicions; establertes i descrites prèviament. Aquests contractes defineixen no només les regles sinó també l'execució d'aquestes, i per tant, el problema de la interpretació subjectiva que teníem als contractes legals no existeix. A més a més, per l'estructura de la xarxa de B*lockchain*, són immutables i totes les modificacions del seu estat queden registrades.

Per tant, i com a resum, podríem dir que tenim un contracte que no necessita cap mena d'autoritat legislativa i/o jurídica en alguns casos concrets. Aquesta propietat no és extensible en general perquè depèn en gran manera de la forma de l'algorisme en si i en última instància de si el consens entre les diverses parts dóna per vàlid, prèviament, el mitjà.

Malgrat que el nom indueix a pensar que els *Smart Contracts* volen ser un símil dels contractes legals, amb tots els avantatges que ofereix l'estructura de l'entorn on es desenvolupen, no només són això. Tot i que els llenguatges utilitzats per a la seva codificació encara són rudimentaris, no deixen de ser algorismes i ens permeten utilitzar-los per molt més que per substituir els contractes legals.

Així doncs, podem tant automatitzar el pagament del lloguer d'un pis o la venda d'una propietat, com una subhasta o una plataforma d'intercanvi de col·leccionables. Fins i tot podem crear aplicacions ja existents i clàssiques però integrades a una xarxa *Blockchain*.

2.1.7. Aplicacions

La tecnologia *Blockchain* s'utilitza en múltiples àrees i sectors on les transaccions, sigui de manera directa o indirecta, tinguin un paper important. L'exemple i l'aplicació més significativa són les criptomonedes. També es pot utilitzar per al registre de documents o propietats, o com un element substitutiu o de millora per a qualsevol empresa.

2.1.7.1. Criptomonedes:

Les criptomonedes són monedes virtuals. Es tracta d'un registre històric de transaccions on es guarda el saldo de cada usuari i els intercanvis realitzats (d'aquí prové el nom de *distributed ledger*). Gràcies a l'estructura de les xarxes *Blockchain*, la criptografia i el fet que sigui un registre distribuït, es pot garantir la seva integritat, controlar la creació d'unitats addicionals i verificar totes les transferències en qualsevol moment per a tots els actors.

2.1.7.2. Registre de documents/propietats:

Les xarxes *Blockchain* generalment són distribuïdes i cada usuari ha de sincronitzar la cadena sencera per garantir-ne la integritat i validesa. Per tant, i per temes d'escalabilitat, no és recomanable que les transaccions continguin grans quantitats de bytes. Fins i tot, en el cas de les xarxes B*lockchain* amb suport per a *Smart Contracts* és poc recomanable.

La solució que s'utilitza actualment per crear registres de documents és obtenir un *hash* del document (o fitxer) desitjat, i guardar-ho en una transacció o *Smart Contract*, en lloc d'emmagatzemar el document sencer. Per la construcció i per

les propietats de les xarxes *Blockchain*, i de la mateixa manera que en les criptomonedes, aquest sistema garanteix la veracitat, existència i integritat d'un document qualsevol.

L'ús de xarxes *Blockchain* per a registrar documents pot complementar i millorar qualsevol registre públic (ex. registre civil, registre de la propietat, registre fiscal …) i disminuir el risc de falta de confiança en figures com jutges, registradors i notaris.

2.1.7.2. Bancs i operacions financeres:

La utilització de xarxes *Blockchain* permet als bancs processar els pagaments de manera més ràpida i precisa. També, els pot permetre una reducció de costos disminuint la quantitat d'intermediaris. Els bancs volen implementar sistemes que disminueixin la quantitat de participants en transaccions, millorant-ne la integritat i la seguretat.

2.1.7.2. Altres usos:

L'ús de les xarxes *Blockchain* s'està expandint a infinits sectors. S'utilitza des del sector sanitari (per emmagatzemar tots els registres mèdics dins la *Blockchain*), fins al sector polític (per realitzar recomptes de vots i així evitar qualsevol frau en unes eleccions polítiques).

Fins i tot, indústries com les dels videojocs estan començant a utilitzar xarxes *Blockchain*. S'ha creat un joc *online* (*Spell of Genesis*) on els jugadors poden intercanviar cartes i criptomonedes del joc utilitzant aquest tipus de xarxes.

També, es pot utilitzar les xarxes *Blockchain* per enviar donacions a *ONGs*, ja que en eliminar l'intermediari, ens assegurem que la donació arriba completament a aquestes entitats.

Aquests són alguns usos on s'utilitza una xarxa *Blockchain*, però les seves aplicacions es poden expandir a tots els sectors.

2.2. *Ethereum*

2.2.1. Introducció

Ethereum és una plataforma de *Blockchain* de codi obert. Es tracta d'una xarxa de computació distribuïda, que a diferència d'altres, es distingeix per poder implementar i executar algorismes de forma nativa sobre els nodes públics que la conformen. Això és possible mitjançant una màquina virtual, l'*Ethereum Virtual Machine* (*EVM*) de tipus *Turing* complet, integrada directament al nucli de la plataforma.

El *Turing* complet integrat a la tecnologia *Blockchain* es refereix a la capacitat que té un llenguatge a poder aplicar-se per resoldre qualsevol problema computacional i implementar estructures complexes.

La *Ethereum Virtual Machine* (*EVM*) és un software que es centra en proporcionar seguretat i executar codis no confiables en ordinadors de tot el món. També, té com a objectiu prevenir els atacs de denegació de servei. Aquesta màquina virtual s'executa cada vegada que s'introdueix informació a la *Blockchain Ethereum* amb la finalitat d'executar el codi a l'ordinador i protegir la informació que es vol introduir.

L'execució d'aquests algorismes té associat un mecanisme de preus basat en el cost computacional que necessita cada transacció. Aquest cost computacional s'anomena gas i es paga utilitzant la moneda principal d'*Ethereum*, *l'Ether*. Aquest preu és variable i s'ha d'especificar i escollir a cada transacció que es duu a terme.

La xarxa *Ethereum* ofereix els següents beneficis:

- Protecció de la informació personal de cada usuari *d'Ethereum*.
- Protecció sobre possibles atacs de hackers gràcies a l'estructura distribuïda de la xarxa.
- Eliminació d'intermediaris permetent interactuar els usuaris entre ells.
- Possibilitat de desenvolupar qualsevol aplicació sobre aquesta xarxa Blockchain.

Una vegada hem introduït els conceptes bàsics i hem explicat els beneficis *d'Ethereum*, ens centrarem en l'estructura d'una transacció, d'un bloc i de l'estat en general de tota la cadena de blocs en aquesta xarxa.

2.2.2. *World State*

Un *World State* és un *mapping* entre adreces i estats dels comptes. Aquest mapping ha estat implementat utilitzant un *Merkle Patricia tree*.

Un *Merkle Patricia Tree* és una estructura en forma d'arbre que intenta proporcionar una estructura de dades autenticades mitjançant criptografia.

Aquesta estructura permet que el node arrel sigui criptogràficament dependent de les dades internes. A més, proporciona immutabilitat, ja que permet recuperar qualsevol estat anterior.

2.2.2.1. Estats dels comptes:

En la B*lockchain Ethereum* existeixen dos tipus de comptes:

- Els comptes de contractes: És un compte únic i immutable. Es crea un hash amb *Keccak-256* a partir d'un *RPL* codificat de l'adreça i del *nonce* de l'usuari.

- Els comptes simples: És un compte únic i immutable. Es calcula a partir d'una clau pública i, per tant, deriva d'una clau privada. Donada una clau privada, s'utilitza un algoritme de signatura digital anomenat *ECDSA*.

ECDSA es tracta d'un esquema per realitzar firmes digitals que utilitza una mida menor i ofereix la mateixa seguretat que *RSA*.

Amb aquest esquema, es genera la seva clau pública (64 Bytes) i, finalment, utilitzant els 40 últims bytes de la clau pública i l'algoritme *Keccak-256* s'obté l'adreça del compte.

La *Ethereum Virtual Machine* tracta d'una forma semblant els dos tipus de comptes. És per aquesta raó que els dos tipus de comptes coinxideixen en els següents camps:

- *Nonce*: És el número de transaccions realitzades per la direcció.
- *Balance*: És la quantitat *d'Ethers* que posseeix la direcció.
- *Storage root*: És el *hash* de 256 bits del node arrel del *Merkle Patricia Tree,* que codifica l'emmagatzematge del compte.
- *Code hash*: És el *hash* del codi *Ethereum Virtual Machine* d'aquest compte. És immutable i no es pot canviar després de la seva creació.

2.2.3. Transaccions

Una transacció a *Ethereum*, és un paquet de dades firmades que contenen un missatge enviat des d'un compte a un altre. Hi ha 2 tipus de transaccions dins *d'Ethereum*:

- Transaccions resultat d'un missatge de petició.
- Transaccions de creació de contractes.

Els dos tipus de transaccions estan formats per uns camps comuns:

- *Nonce*: És el nombre de transaccions que emet l'emissor.
- *GasPrice*: És el número de gas que es pagarà per cada unitat per de gas.
- *GasLimit*: És el valor màxim de gas que s'utilitza per generar aquella transacció.
- *To*: És l'adreça de 160 bits del destinatari de la transacció de missatge o de creació de contracte.
- *Value*: És un valor igual al nombre de *Wei's*.
- v, r, s: Són els valors corresponents a un contracte ja creat.

A més a més, una transacció feta a partir d'una petició conté el següent camp:

- *Data*: És una matriu de bytes de mida il·limitada que especifica el codi *EVM* per al procediment d'inicialització del compte. És un fragment del codi *EVM*, i retorna el cos.

Finalment, una transacció de creació de contracte conté el següent camp:

- *Init*: És una matriu de bytes de mida il·limitada que especifica les dades d'entrada de la petició del missatge.

2.2.4. Blocs

Els blocs *d'Ethereum* estan formats per la unió entre la informació de la capçalera del bloc, la informació corresponent a les transaccions i un conjunt d'altres capçaleres de blocs anteriors que tenen un bloc pare igual al bloc pare del bloc que estem analitzant (coneguts com a *ommers*).

La capçalera del bloc conté la següent informació:

- *ParentHash*: És el *hash Keccak* de 256 bits de l'encapçalat del bloc principal.
- *OmmersHash*: És el *hash Keccak* de 256 bits de la part de la llista d'ommers d'aquest bloc.
- *Beneficiary*: És l'adreça de 160 bits a la qual es transfereixen totes les tarifes recollides de la mineria d'èxit d'aquest bloc.
- *StateRoot*: És el hash Keccak de 256 bits del node arrel del *trie* d'estat, una vegada que totes les transaccions s'executen i s'apliquen les finalitzacions.
- *TransactionsRoot*: És el *hash Keccak* de 256 bits del node arrel de l'estructura *trie* on s'inclou cada transacció en la part de la llista de transaccions del bloc.
- *ReceiptsRoot*: És el *hash Keccak* de 256 bits del node arrel de l'estructura *trie* on s'inclou els rebuts de cada transacció a la part de la llista de transaccions del bloc.
- *LogsBloom*: És el filtre Bloom compost per informació indexable, continguts en cada entrada del registre des de la recepció de cada transacció a la llista de transaccions.
- *Difficulty*: És un valor corresponent al nivell de dificultat d'aquest bloc. Aquest valor es pot calcular a partir del nivell de dificultat del bloc anterior i la marca de temps.
- *Number*: És un valor igual al nombre de blocs ancestres. El bloc gènesi té un *number* igual a cero.
- *GasLimit*: És un valor igual al límit actual de la despesa de gas per bloc.
- *GasUsed*: És un valor igual al total del gas utilitzat en les operacions d'aquest bloc.
- *Timestamp*: És un valor igual al valor del temps Unix en l'inici del bloc.
- *ExtraData*: És una matriu de bytes que conté dades rellevants per a aquests bloc. Aquest valor ha de ser màxim de 32 bytes.
- *MixHash*: És un *hash* de 256 bits que, juntament amb el *nonce*, demostra que s'ha realitzat una quantitat suficient de computació en aquest bloc.
- *Nonce*: És un valor de 64 bits que, combinat amb el *mixHash*, demostra que s'ha realitzat una quantitat suficient de computació en aquest bloc.

Els altres dos components del bloc no només són una llista d'encapçalaments, incorporen la informació de les transaccions i referències a blocs anteriors.

2.2.5. Aplicacions

Les aplicacions *d'Ethereum* es realitzen a partir de *Smarts Contracts. Ethereum* permet als desenvolupadors construir i implementar *DApps* (apartat 2.3. DApps). Al desenvolupar qualsevol tipus de *DApps*, podem dir que Ethereum es pot implementar en tants àmbits diferents com diferents idees tingui un desenvolupador.

2.2.6. *Tokens*

Un *token* es pot definir com la representació de qualsevol cosa. Més específicament, un *token* pot representar monedes, punts o qualsevol mesura que li doni valor a alguna cosa. D'aquesta manera, un *token* no només podria substituir una moneda, si no podria substituir qualsevol element que representi el valor d'una moneda, com uns cupons, vals, escriptures, butlletes, etc.

En termes de *Blockchain*, un *token* no deixa de tenir el mateix significat que la definició anterior. Així doncs, un *token* de *Blockchain* segueix sent una representació d'un actiu, d'un bé, d'uns punts o d'una criptomoneda.

Diferències entre *Tokens* i Criptomonedes:

Una criptomoneda no deixa de ser una moneda digital que està distribuïda en una *Blockchain* i que només té la funció monetària. En canvi, els *tokens* donen l'oportunitat d'utilitzar-los per totes les funcions anteriorment explicades.

Un *token*, normalment, està creat per sobre d'una *Blockchain* i utilitza la tecnologia de la cadena a la qual s'introdueix. Això significa que té un procés de creació molt més fàcil i ràpid, ja que no necessita la modificació del codi ni la creació d'una nova *Blockchain*.

En canvi, les criptomonedes tenen una estructura pròpia. Cada criptomoneda utilitza la seva pròpia *Blockchain*.

Tipus de *Tokens*:

Per la creació de *tokens* s'acostuma a utilitzar uns *Smarts Contracts* programats per gestionar tant els balanços com les transaccions d'aquests *tokens*.

A *Ethereum* existeixen diferents estàndards que apliquen normes per a diferents tipus de *tokens*. Aquests estàndards s'anomenen *ERC* (*Ethereum Request for Comment*) i són normes i consensos per la creació d'aquests *tokens*. A més, s'inclou l'*Smart Contract* al qual s'està fent referència. Aquests estàndards els pot crear qualsevol persona, però necessita el consens i suport

de la comunitat, i finalment una revisió per part de la comunitat de desenvolupadors per tal que es creï la norma.

Existeixen molts tipus de *tokens*, però els més utilitzats són els següents:

- *ERC-20*: És l'estàndard més famós i utilitzat. Defineix una sèrie de regles senzilles les quals descriuen com es transfereix els actius entre les adreces. Això permet crear una nova comptabilitat paral·lelament i dintre de la mateixa xarxa *Ethereum* però creant una nova moneda.
- *ERC-223*: Es tracta d'una proposta d'estàndard per tal d'evitar qualsevol pèrdua de *tokens* en el moment de realitzar una transferència. És un estàndard similar a l'ERC-20 però inclou una funcionalitat que garanteix que els *tokens* no s'envien a cap contracte que no estigui preparat per complir les normes establertes.
- *ERC-721*: Aquest estàndard defineix uns *tokens* no fungibles. A diferència dels anteriors *tokens* que són fungibles (cada *token* és el mateix que qualsevol altre) cada *token* ERC-721 és únic.
- *ERC-777*: És un estàndard que té com a objectiu principal intentar solucionar les limitacions de l'estàndard ERC-20. Les funcions principals d'aquest estàndard és permetre els enviaments de *tokens* en nom d'una altra adreça i enviar o rebre controls sobre una sèrie de *tokens*.

2.2.7. *Tokens* ERC-20

Com s'ha introduït anteriorment, els *Tokens ERC-20* són els *tokens* més utilitzats. Aquest tipus de *tokens* no deixen de ser un *Smart Contract* que defineix una estructura de dades ja preestablerta.

L'objectiu principal dels *tokens* ERC-20 és construir una sèrie de normes per tal de definir la gestió de balanços de cada adreça i l'administració de les transferències de *tokens*. A més, permeten que qualsevol *token* es pugui utilitzar en qualsevol aplicació que admeti aquest tipus de *tokens*.

Les característiques principals d'aquests *tokens* són les següents:

- Són reconeixibles i diferenciables, ja que tenen un nom i un símbol que els identifica.
- Són capaços gestionar i controlar els balanços de cada adreça.
- Permeten realitzar transferències de tokens de manera senzilla.
- Tenen la capacitat de controlar les transferències utilitzant mètodes d'aprovació i de concessió de permisos.

La interfície que es defineix els *Tokens* ERC-20 és la següent:

```
pragma solidity >= 0.4.0 <0.6.0;

interface IERC20 {

    // METADATA FUNCTIONS
    function name() external pure returns (string memory _name);
    function symbol() external pure returns (string memory _symbol);

    function totalSupply() external view returns (uint);
    function balanceOf(address tokenOwner) external view returns (uint balance);
    function allowance(address tokenOwner, address spender) external view returns (uint remaining);
    function transfer(address to, uint tokens) external returns (bool success);
    function approve(address spender, uint tokens) external returns (bool success);
    function transferFrom(address from, address to, uint tokens) external returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

Fig. 2.1 Interfície *ERC20.sol*

Mètodes:

- *name*(): Funció amb la qual s'estableix en nom al *token* que es crea. S'utilitza per diferenciar el *token* creat amb els altres *tokens* ja existents.
- *symbol*(): Funció opcional amb la qual s'estableix una abreviació o símbol al token que es crea.
- *decimal*(): Funció opcional amb la qual es defineix el nombre de decimals que tindrà el *token*.
- *totalSupply*(): Funció que retorna el nombre total de *tokens* que existeixen.
- *balanceOf*(): Funció que retorna el nombre de *tokens* d'una adreça donada com a paràmetre.
- *allowance*(): Funció que retorna el nombre de *tokens* que es pot retirar d'una adreça donada com a paràmetre.
- *transfer*(): Funció amb la qual es transfereixen *tokens* des de la pròpia compta a una adreça donada com a paràmetre.
- *approve*(): Funció amb la qual es permet la retirada de *tokens* d'una adreça donada com a paràmetre.
- *transferFrom*(): Funció amb la qual es transfereixen *tokens* des d'una compta a una altra. Les dues adreces són donades com a paràmetre.

Events:

- *Transfer*(): *Event* que és llençat en el moment que es transfereixen *tokens*.
- *Approval*(): *Event* que és llençat en el moment que s'aprova la transferència de *tokens*.

2.3. *DApps*

L'abreviació *DApp* es refereix a aplicació descentralitzada/distribuïda. Una *DApp* és una aplicació que s'executa i/o és suportada sobre una xarxa *Blockchain*.

Per executar una *DApp*, es necessita que aquesta es connecti a un client *Blockchain* (encarregat de gestionar la B*lockchain* i, en el cas *d'Ethereum*, l'*Ethereum Virtual Machine*) mitjançant una *api*/llibreria per gestionar tot el que s'introdueix o es consulta a la *Blockchain*.

La llibreria més famosa i més utilitzada per desenvolupar *DApps* sobre *Ethereum* és la llibreria *Web3*. *Web3*.js és una API dissenyada per la interacció amb la xarxa Blockchain *d'Ethereum*. Ha estat creada principalment per utilitzar-ho amb *Javascript*, però també s'ha creat pels següents llenguatges: *Python*, *Java*, *Purescript*, *Php*, etc…

2.3.1. Tipus de *DApps*

Existeixen 3 diferents tipus de *DApps* segons la *Blockchain* que utilitzin:

- Tipus 1: Són les aplicacions descentralitzades que tenen la seva pròpia *Blockchain*. *Ethereum*, *Bitcoin*, i totes les altres criptomonedes entrarien dins d'aquest tipus de *DApps*.
- Tipus 2: Són les aplicacions descentralitzades que utilitzen alguna *Blockchain* del Tipus 1 en lloc d'utilitzar la seva pròpia. Aquestes *DApps* són protocols que funcionen a partir dels seus *tokens* o a partir dels *tokens* de la *Blockchain* en la que operen.
- Tipus 3: Són les aplicacions descentralitzades que utilitzen alguna *Blockchain* del Tipus 2 en lloc d'utilitzar la seva pròpia. Igual que les *DApps* de Tipus 2, poden utilitzar els seus *tokens* o els dels *tokens* de la *Blockchain* en la que operen.

2.3.2. Diferència entre *App* i *Dapp*

La diferència principal entre una *App* i una *DApp* és que les *DApp* necessiten estar connectades a un client de *Blockchain* a través d'una API per poder interaccionar amb una xarxa *Blockchain*. A diferència de les Apps corrents, on el codi del *Back-End* està corrent sobre un servidor centralitzat, les *DApps* executen el codi *Back-End* en *Smart Contracts* sobre una xarxa *Blockchain*.

Una altra diferència entre les *Apps* i les *DApps* és que aquestes últimes necessiten un pagament cada vegada que es vulgui interaccionar amb la xarxa *Blockchain*, en canvi, existeixen moltíssimes Apps convencionals gratuïtes. Però, el fet de realitzar un pagament per cada transacció a la *Blockchain* permet que una *DApp* guardi una quantitat *d'Ethers* i es distribueix segons els requeriments de l'aplicació. L'exemple d'una loteria on tots els participants paguen una quantitat *d'Ethers* i aquesta quantitat s'assigna al guanyador seria l'exemple més clar.

Finalment, una de les diferències més notables és que la seguretat de les *DApps* és, normalment, molt més important que les *Apps* centralitzades, ja que adopten totes les característiques de seguretat i criptografia de les xarxes *Blockchain*.

CAPÍTOL 3. ARQUITECTURA DE DESENVOLUPAMENT

En aquest capítol s'exposa l'arquitectura de l'aplicació així com les tecnologies que s'han utilitzat per desenvolupar-la. S'explica l'estructura que té el projecte desenvolupat, així com una introducció de les parts del projecte i una àmplia explicació de les tecnologies utilitzades per desenvolupar-lo i testejar-lo.

3.1. Arquitectura

El projecte desenvolupat s'estructura de la següent manera:

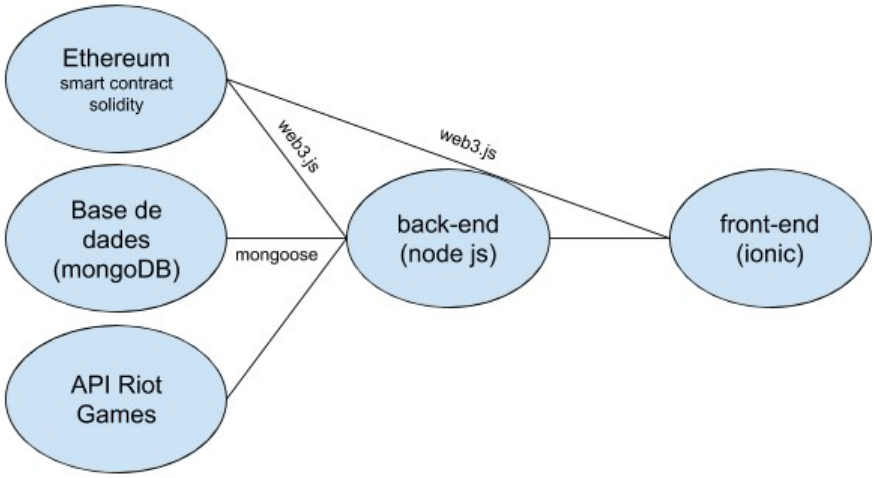


Fig. 3.1 Diagrama de l'estructura del projecte

Aquest projecte conté un servidor (*Back-End*) desenvolupat amb Node JS que gestiona totes les dades agafades des de la *Blockchain*, la base de dades i la *API* de *Riot Games.* A més, exposa una *API* per proporcionar tota la informació per mostrar a l'aplicació (*Front-End*).

- *Smart contract:* El *Smart Contract* s'ha realitzat amb *Solidity* i inclou la creació i les gestions del *token ERC20* que s'ha creat. A més a més, es gestionen les apostes realitzades.
- Base de dades: S'utilitza una base de dades MongoDB, on s'emmagatzema tota la informació dels usuaris, apostes realitzades i informació de partides ja extretes de l'*API* de *Riot Games.*
- *API Riot Games*: *API* d'on s'extreu la informació dels *summoners* i de les partides en directe buscades.
- Servidor: El *Back-End* s'ha realitzat amb *Node JS*. Executa la connexió amb totes les altres parts del projecte i gestiona tota la informació tant d'usuaris, com d'apostes. A més, realitza totes les funcions per gestionar la *wallet*.

- Aplicació: L'aplicació s'ha realitzat amb *Ionic* 4. Mostra tota la informació d'usuaris i d'apostes, i permet gestionar els *tokens* de cada usuari i realitzar les apostes que es desitgen. A més, realitza totes les funcions de *wallet* que només s'utilitzen al *Front-End*.

3.2. Tecnologies utilitzades

Les tecnologies o softwares utilitzats es poden diferenciar segons el principal propòsit pel qual s'utilitzen.

En primer lloc trobem els clients per executar *Blockchains*. Aquests softwares s'utilitzen com a pont per poder sincronitzar i realitzar qualsevol connexió amb la *Blockchain* indicada. Existeixen softwares com *Geth*, que serveixen com a node d'una *Blockchain* pública *d'Ethereum*, o altres com *Ganache*, que serveixen com a *Blockchain* personal pel desenvolupament en xarxes *Ethereum*.

A continuació, trobem els llenguatges de programació utilitzats per desenvolupar el projecte. Per programar els *Smarts Contracts* s'utilitza el llenguatge *Solidity* i per desenvolupar l'aplicació es segueix la pila de frameworks *MEAN* (*MongoDB, Express, Angular/Ionic, Node JS*). Aquest grup de *frameworks* estan basats en el llenguatge de programació *Javascript*.

Després, s'utilitzen diferents llibreries o *frameworks* per realitzar diferents tests. En primer lloc, trobem *Truffle* que no només serveix per provar tots els S*marts Contracts* desenvolupats, sinó que a més dóna un munt de funcions que faciliten la feina dels programadors. També, s'utilitzen les llibreries *Chai* i *Mocha* per realitzar els tests de cada funció realitzada.

Finalment, trobem *Apache Cordova* que es tracta d'un software que permet crear una aplicació nativa per qualsevol plataforma mòbil.

En l'*Annex D: Descripció de les tecnologies* utilitzades es troben descrites totes les tecnologies explicades de forma detallada.

CAPÍTOL 4. *E-bet*

Aquest capítol descriu l'aplicació que s'ha desenvolupat en aquest projecte. S'explica la idea i la descripció de l'aplicació, així com les funcionalitats principals que disposa. A continuació, es descriuen els mecanismes d'incentius i una petita informació empresarial.

4.1. Descripció de l'aplicació

L'aplicació *E-bet* es tracta d'una plataforma on poder interconnectar diferents jugadors *online* mitjançant un joc d'apostes un contra un. Per poder realitzar aquestes apostes, els usuaris han d'adquirir els *tokens E-bet* intercanviables per la criptomoneda *d'Ethereum*, anomenada *Ethers*. En el moment en que l'usuari té un mínim d'un *token* en un dels seus comptes *d'Ethereum* ja pot començar a realitzar una aposta.

L'objectiu principal d'aquest joc és encertar el guanyador d'una partida del videojoc *League of Legends*.

El guanyador d'una aposta aconsegueix la suma d'*E-bet* apostats per ell mateix i la mateixa quantitat apostada per al seu contrincant, que haurà perdut aquests *E-bet*.

A més, en qualsevol moment l'usuari pot vendre els *tokens E-bet* a canvi *d'Ethers*. Així, com més *E-bet* guanyi l'usuari, més *Ethers* podrà obtenir.

A part, l'aplicació *E-bet* segueix el model d'aplicacions de les xarxes socials. Té incorporat un xat per poder dialogar amb els diferents usuaris, així com la funció de seguir un usuari com a amic i un perfil personal per cada usuari per poder observar les seves apostes i les seves estadístiques.

El projecte es troba en els següents *links* de *github*:

- *Back-End*: https://github.com/Aleix11/server-tfg
- *Front-End*: https://github.com/Aleix11/client-tfg-ionic

4.2. Fluxos – Funcionalitats

Les funcions principals d'aquesta aplicació es poden separar entre funcions d'autenticació, funcions de gestió de les apostes i funcions de xarxes socials.

- Les funcions d'autenticació s'inicien amb el registre d'un compte i la posterior creació de la *wallet*. Si un usuari ja ha creat un compte, pot utilitzar la funció d'entrada al compte a partir del nom d'usuari i la contrasenya.
- Les funcions de gestió d'apostes inclouen tant la creació, l'acceptació i el tancament de les apostes.

- Les funcions de xarxes socials són aquelles que permeten connectar els diferents usuaris i fer més agradable l'experiència de l'usuari amb l'aplicació.

4.2.1. Registre

Per poder utilitzar l'aplicació és necessari efectuar un registre. Aquest registre està diferenciat en dues grans parts:

1. Registre del compte: Per realitzar el registre del compte és necessari introduir un nom d'usuari, un *email* i una contrasenya.
2. Creació de la *wallet*: En completar la primera part del registre s'efectuarà la creació automàtica de la *wallet*. A partir d'aquest punt, es requereix crear un compte en la *Blockchain* o importar-lo introduint una clau privada. El següent pas és introduir una contrasenya per la *wallet*, la qual permetrà gestionar la pròpia *wallet* i realitzar les transaccions de creació i acceptació d'apostes.

4.2.2. Creació aposta

En aquest moment, l'usuari 1 ha de crear manualment l'aposta.

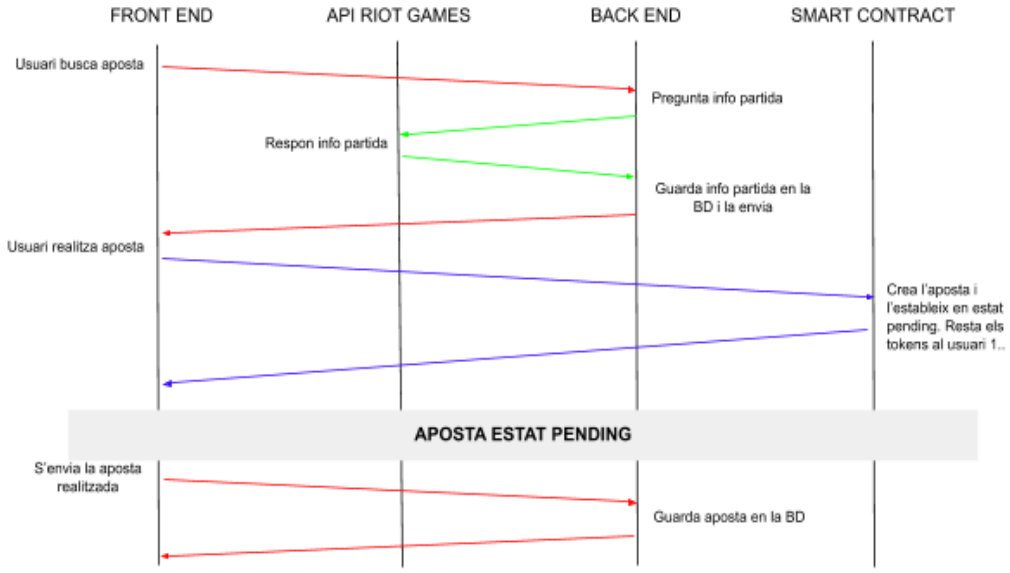


Fig. 4.1 Diagrama de creació d'una aposta

En aquesta situació l'usuari 1 busca un *summoner* i realitza l'aposta en el moment que ell vulgui. Per crear-la, l'usuari 1 realitza una petició *http* a *l'API* del *Back-End* de l'aplicació, i aquest realitza una petició *http* a l'*API* de *Riot Games* i guarda la informació rebuda en la base de dades. En aquest moment, es respon al Front-End donant tota la informació de la partida.

A partir d'aquest punt, l'usuari 1 envia la transacció a la *Blockchain* executant el mètode *betCreate*() del *Smart Contract Bet.sol.*

El *Smart Contract* crea l'aposta i l'estableix en estat *Pending*. A més, actualitza el balanç de l'usuari 1 restant-li els *tokens* apostats.
Finalment, i una vegada s'ha confirmat la transacció de creació de l'aposta, es realitza una petició *http* cap a *l'API* del *Back-End* enviant l'aposta realitzada. El *Back-End* guarda l'aposta en la base de dades.

A continuació, l'usuari 1 ha de seguir els següents passos per tal de realitzar la transacció:

1. Buscar el *summoner* que estigui està jugant una partida: Per crear l'aposta l'usuari ha d'introduir el nom d'aquest *summoner*. En cas negatiu, apareixerà un missatge indicant que aquell *summoner* no està jugant cap partida en aquell moment.
2. Seleccionar l'equip pel qual es vol apostar: Si el *summoner* introduït està jugant una partida, es mostrarà una llista amb tots els *summoners* de cada equip, i posteriorment, un marcador per seleccionar l'equip al qual es vol apostar.
3. Seleccionar la duració de l'aposta: L'usuari creador de l'aposta ha de seleccionar el temps que vol que duri l'aposta. Si passat aquest temps la partida no ha acabat, l'aposta es tancarà i quedarà en pausa fins al moment en què acabi la partida i s'obtingui l'equip guanyador. En el cas que la partida acabi abans del temps de durada de l'aposta, aquesta es tancarà.
4. Seleccionar els *tokens* que es volen apostar: L'usuari creador de l'aposta ha de seleccionar els *tokens* que vol apostar.
5. Crear aposta: Una vegada l'usuari ha introduït totes les dades requerides, es mostra un resum amb tota la informació de l'aposta, dels equips i l'adreça de l'usuari que crea l'aposta.

4.2.3. Acceptació aposta

En aquest punt, l'usuari 2 troba l'aposta mitjançant un buscador.

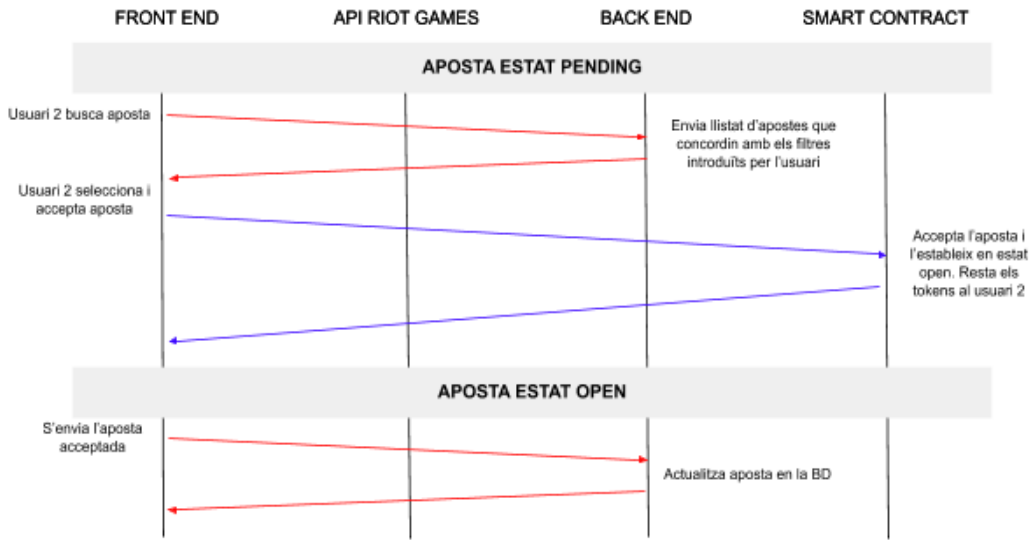


Fig. 4.2 Diagrama d'acceptació d'una aposta

En aquesta situació l'usuari 2 realitza una cerca en el llistat d'apostes en estat *Pending* aplicant una sèrie de filtres. En el moment en què l'usuari 2 troba una aposta que voldria acceptar, clica sobre ella i pot visualitzar tota la informació de la partida i de l'aposta.

Posteriorment, l'usuari 2 accepta l'aposta i envia la transacció a la *Blockchain* executant el mètode *betOpen*() del *Smart Contract Bet.sol.*

El *Smart Contract* estableix l'aposta en l'estat Open. A més, actualitza el balanç de l'usuari 2 restant-li els *tokens* apostats.

4.2.4. Tancament aposta – Recompensa

Es poden donar tres situacions en les quals es produeix el tancament d'una aposta:

1. Ha acabat la duració de l'aposta o ha finalitzat la partida d'una aposta en estat *Open* i hi ha hagut un guanyador:

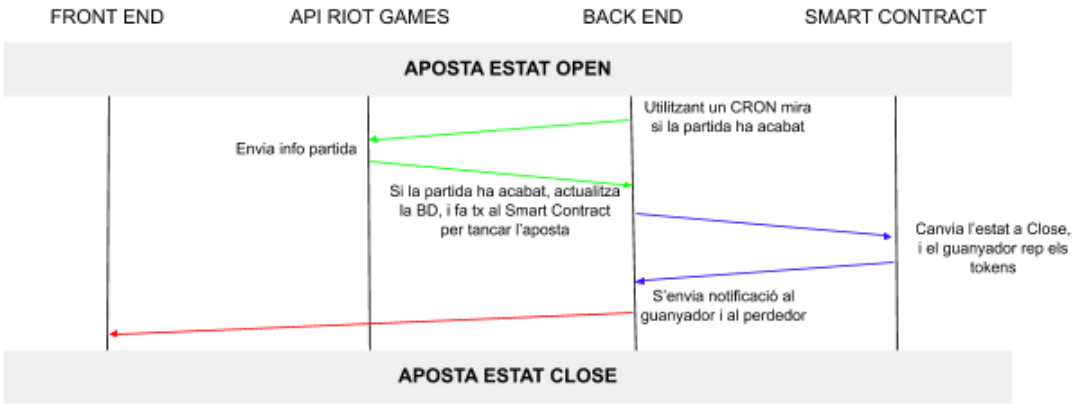


Fig. 4.3 Diagrama de tancament d'una aposta

El CRON que s'executa en el *Back-End* realitza una petició *http* a *l'API* de *Riot Games* per comprovar si la partida ha acabat. En cas afirmatiu, s'actualitza la informació de la partida en la base de dades i el propietari del *Smart Contract* envia la transacció a la *Blockchain* executant el mètode *betClose*() del *Smart Contract Bet.sol.*

El *Smart Contract* canvia l'estat de l'aposta a *Close*, i envia al guanyador la suma de la quantitat de *tokens* apostada per cada usuari.

Al confirmar-se la transacció, es modifica la informació de l'aposta a la base de dades i s'envia una notificació a l'usuari guanyador i perdedor informant del resultat de la partida i del guanyador de l'aposta.

2. Ha acabat la duració de l'aposta o ha acabat la partida d'una aposta en estat *Pending*:

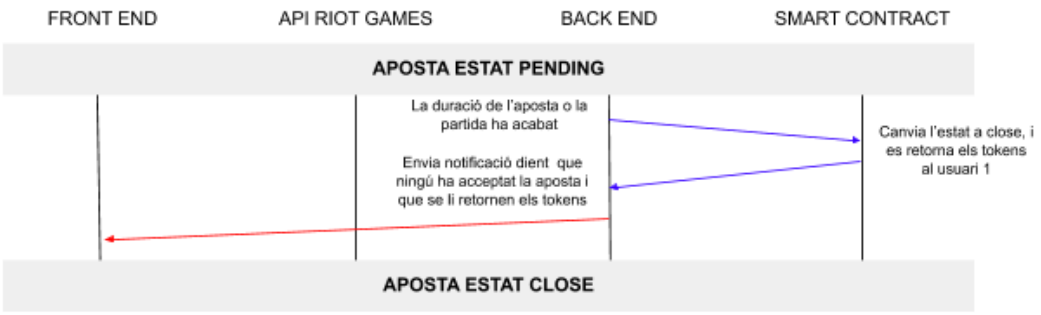


Fig. 4.4 Diagrama de tancament d'una aposta sense guanyador

El *CRON* que s'executa en el *Back-End* comprova que una partida o una aposta en estat *Pending* hagi acabat. En cas afirmatiu, el propietari del *Smart Contract* envia la transacció a la *Blockchain* executant el mètode *betCloseFromPending*() del *Smart Contract Bet.sol.*

El *Smart Contract* canvia l'estat de l'aposta a *Close*, i retorna els *tokens* apostats a l'usuari 1.

Al confirmar-se la transacció, es modifica la informació de l'aposta en la base de dades i s'envia una notificació a l'usuari 1 informant que ningú ha acceptat l'aposta, que s'ha modificat el seu estat a *Close* i que se li han retornat els *tokens*.

3. Ha acabat la duració de l'aposta o ha acabat la partida d'una aposta en estat *Open* i no hi ha hagut guanyador (s'ha produït *remake*):

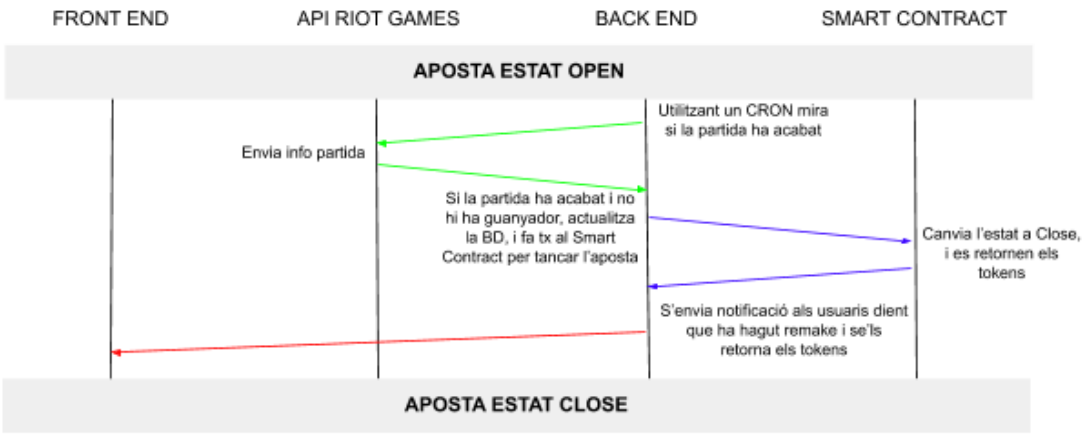


Fig. 4.5 Diagrama de tancament d'una aposta amb remake

El *CRON* que s'executa en el *Back-End* realitza una petició *http* a *l'API* de *Riot Games* per comprovar si la partida ha acabat. En cas afirmatiu, comprova que no ha hagut guanyador (hi ha hagut r*emake*) s'actualitza la informació de la partida en la base de dades i el propietari del *Smart Contract* envia la transacció a la *Blockchain* executant el mètode *betCloseRemake*() del *Smart Contract Bet.sol.*

El *Smart Contract* canvia l'estat de l'aposta a *Close*, i retorna als dos usuaris els *tokens* apostats.

En confirmar-se la transacció, es modifica la informació de l'aposta en la base de dades i s'envia una notificació al dos usuaris informant-los que hi ha hagut *remake* i que se'ls retorna els *tokens* apostats.

En l'*Annex E: Fluxos – Funcionalitats futures* es troben descrites unes funcions per crear i acceptar apostes mitjançant notificacions.

4.2.5. Compra-Venda de *tokens E-bet*

Per poder realitzar qualsevol funció anterior, és necessari disposar de *tokens E-bet* en l'adreça on s'executen les transaccions. Per poder obtenir aquests *tokens* s'han d'intercanviar per *Ethers*.

L'intercanvi d'*Ethers* i *tokens E-bet* ha estat definit de la següent manera:

- Si l'usuari paga 1 *Ether*, obté 900 *E-bet.*
- Si l'usuari paga 1000 *E-bet,* obté 1 *Ether*.

Per realitzar la compra de *tokens E-bet*, s'executen els següents passos:

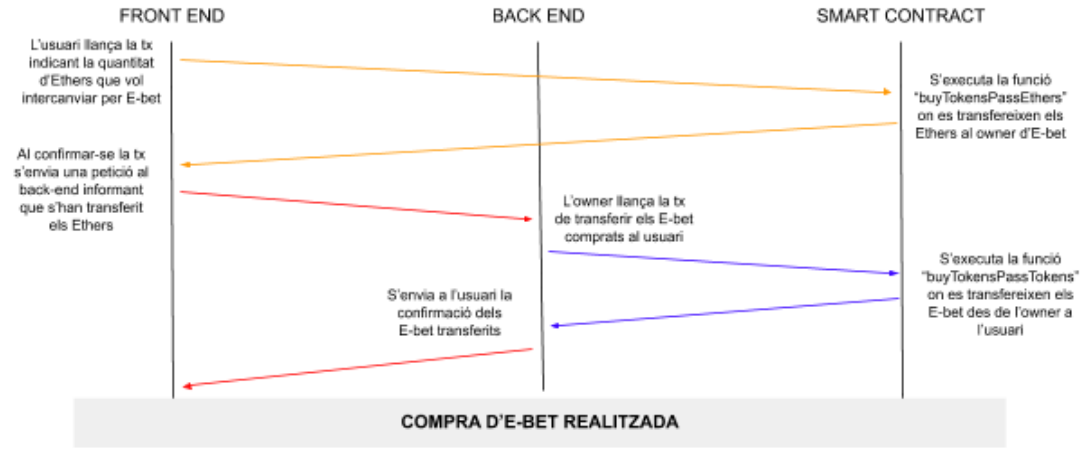


Fig. 4.6 Diagrama de la compra de tokens E-bet

L'usuari llança una transacció executant el mètode *buyTokensPassEthers*() del *Smart Contract Bet.sol* indicant el nombre *d'Ethers* que vol intercanviar per *tokens E-bet*. Aquest mètode transfereix els *Ethers* enviats per l'usuari a *l'owner* del contracte.

En confirmar-se la transacció, l'usuari envia una petició *http* al *Back-End* informant que ha realitzat l'anterior transacció. A continuació, *l'owner* llança una transacció executant el mètode *buyTokensPassTokens*() on envia el *tokens E-bet* a l'usuari.

Per realitzar la venda de *tokens E-bet*, s'executen els següents passos:

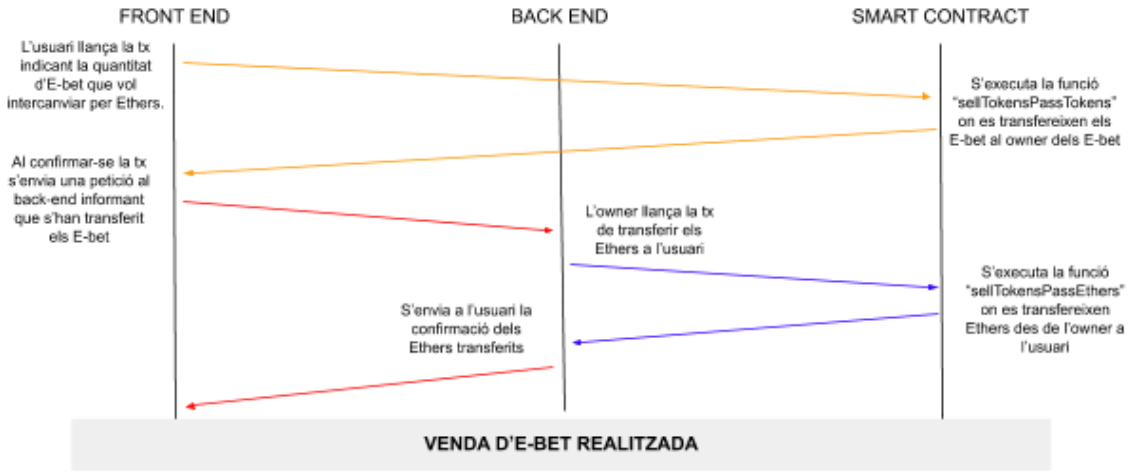


Fig. 4.7 Diagrama de la venda de tokens E-bet

L'usuari llança una transacció executant el mètode *sellTokensPassTokens*() del *Smart Contract Bet.sol* indicant el nombre d'*E-bet* que vol intercambiar per Ethers. Aquest mètode transfereix els *E-bet* enviats per l'usuari a *l'owner* del contracte.

Al confirmar-se la transacció, l'usuari envia una petició *http* al *Back-End* informant que ha realitzat l'anterior transacció. A continuació, *l'owner* llança una transacció executant el mètode *sellTokensPassEthers*() on envia els *Ethers* a l'usuari.

4.2.6. Funcions Xarxes Socials

Una de les idees de l'aplicació, com s'ha comentat anteriorment, és connectar gent per poder fer més emocionants aquest tipus de partides realitzant apostes. Per aquest motiu, s'han introduït una sèrie de funcions per tal que els diferents usuaris es connectin entre ells i es creïn i s'acceptin més apostes:

1. Perfil: Cada usuari té un perfil on es veurà la seva foto, el seu nom d'usuari, la quantitat d'amics que té, el *summoner* preferit escollit, el llistat d'apostes realitzades i diferents estadístiques. Aquesta funció permet que cada usuari es mostri davant dels altres i, a més, que el mateix usuari pugui conèixer les seves estadístiques d'apostes realitzades.

2. Afegiment d'amics: S'ha implementat la funció on cada usuari pot afegir altres usuaris a una llista d'amics. En crear aquesta llista cada usuari té més fàcil poder accedir al perfil de qualsevol amic i contactar d'una manera més ràpida.
3. Xat: Cada usuari podrà xatejar amb qualsevol altre gràcies a aquesta funció. Aquest fet fa que cada usuari pugui contactar amb altres persones que no conegui i poder interactuar per crear una aposta futura.

4.3. Mecanismes d'incentius

Per poder incentivar l'ús de l'aplicació, en un futur es poden instaurar les següents funcions:

- Regalar *E-bet en* registrar-se: Per incentivar el registre d'usuaris en l'aplicació i permetre que aquests realitzin apostes a l'inici sense haver de comprar els seus primers *tokens E-bet*, se'ls regalarà un nombre baix de *tokens*. D'aquesta manera tots els usuaris podran realitzar la seva primera aposta sense haver de gastar-se ni un *Ether*.
- Regalar *E-bet* en portar un amic: Per incentivar l'ús i l'expansió de l'aplicació, es regalaran *tokens E-bet* si un usuari fa que un altre usuari es registri en l'aplicació. Cada usuari tindrà un codi personal que, si un usuari l'introdueix en registrar-se, es regalaran *token E-bet* extres tant per l'usuari que porta l'amic com l'amic que es registri. A més, per tal d'evitar la creació de comptes falsos o bots, s'haurà de realitzar una confirmació per mitjà d'un telèfon mòbil.

4.4. Model de negoci – Viabilitat econòmica

Per tal d'assegurar-nos la viabilitat econòmica i determinar si el projecte desenvolupat es pot realitzar en termes de negoci, s'ha realitzat el següent model de negoci.

4.4.1. Segment de clients

El projecte desenvolupat va dirigit a tots els jugadors i/o aficionats als esports electronics. Però, hi ha la limitació que tots els usuaris de l'aplicació han de ser majors d'edat per tal que puguin realitzar una aposta.

4.4.2. Proposta de valor

Com s'ha comentat amb anterioritat, l'aplicació *E-bet* permet realitzar apostes un contra un del videojoc *League of Legends* utilitzant el t*oken ERC20 E-bet*.

La proposta de valor d'aquest projecte és permetre als usuaris no només realitzar apostes en partides professionals d'aquest *e-sport,* sinó permetre realitzar una aposta de qualsevol partida de *League of Legends* i de qualsevol usuari sense dependre del nivell assolit.

4.4.3. Relacions amb els clients

Per tal d'impulsar el registre d'usuaris en l'aplicació i cuidar la relació amb els clients, s'han proposat els mecanismes d'incentius explicats en l'apartat 5.3. Mecanismes d'incentius. En ells, es regalen un nombre determinat de *tokens E-bet* per impulsar als usuaris a crear i acceptar més apostes.

4.4.4. Canals de distribució

En realitzar una aplicació híbrida, l'aplicació desenvolupada es podrà distribuir tant per telèfons amb sistema operatiu *Android*, *iOS* o *Windows Phone*. D'aquesta manera, qualsevol persona que vulgui utilitzar l'aplicació només necessita un telèfon mòbil intel·ligent.

4.4.5. Fonts d'ingressos – Viabilitat econòmica

En el moment de realitzar el desenvolupament de l'aplicació s'han estudiat quines són les millors fonts d'ingressos.

4.4.5.1. Publicitat – Google Ads

La font d'ingrés principal per aquesta aplicació serà la publicitat. En un futur, i mitjançant l'ús de *Google Ads*, es podrà incrustar anuncis en l'aplicació de manera senzilla i de forma elegant. A més, *Ionic* posa en disposició dels desenvolupadors el *plugin AdMob Pro*.

AdMob Pro es un *plugin* per mostrar anuncis dissenyats utilitzant-ho en aplicacions híbrides basades en *HTML5*. Existeixen els següents tipus d'anuncis:

- *Banner*: Anunci petit incrustat normalment en la part inferior de la pantalla. És un tipus d'anunci que permet als usuaris seguir gaudint de la visió de gairebé tota la pantalla. És el tipus d'anunci que menys beneficis aporta. Es podria introduir en qualsevol pantalla de la nostra aplicació.
- *Interstitial*: Es tracta d'un anunci en format imatge, text o vídeo en pantalla completa. Aporten molts més beneficis que els anuncis *Banner*, però es perd la visió completa de la pantalla. Aquest tipus d'anuncis es podrien llençar en el moment que un usuari crea o accepta una aposta.
- *Reward Video*: Són els vídeos que utilitzen molts jocs de mòbil per tal de guanyar una recompensa. Aporten un gran benefici, però per la jugabilitat de l'aplicació desenvolupada no es podrien implementar.

4.4.5.2. Comissions

Un altra font d'ingressos que podem trobar en la nostra aplicació es tracta de les comissions que s'apliquen a l'hora d'establir una aposta en estat *Close*.

En el moment d'enviar els *tokens* guanyats al guanyador de l'aposta, aquest rebrà el 95% del *tokens* apostats. El 5% de *tokens E-bet* restant se'ls quedarà *l'owner*.

El mateix fet succeeix en el moment que un usuari ha creat una aposta i ningú l'ha acceptat. Quan es llença la transacció de retorn dels *tokens E-bet*, el creador només rep el 95% dels *tokens* apostats, i el sobrant se'ls queda *l'owner*.

Introduint aquesta comissió no només s'obtenen beneficis, sinó també es cobreixen les despeses de gas al realitzar qualsevol de les dues transaccions anteriorment descrites.

4.4.5.3. Compra-venda de tokens E-bet

L'última font d'ingressos que podem observar en el nostre projecte és la compra-venda de *tokens E-bet.* S'ha establert un valor fixe per realitzar aquests intercanvis de *tokens E-bet* per *Ethers.*

Si un usuari compra *tokens E-bet* per valor d'un *Ether*, rebrá 900 *E-bet*.

En canvi, si un usuari vol vendre *tokens E-bet* per valor d'un *Ether*, haurà d'abonar 1000 *E-bet*.

Introduint aquesta diferència de preu entre la compra i la venda de *tokens E-bet* no només s'obtenen beneficis, sinó també es cobreixen les despeses de gas al realitzar qualsevol de les dues transaccions anteriorment descrites.

CAPÍTOL 5. *SMART CONTRACTS*

En aquest capítol s'expliquen els *Smart Contracts* desenvolupats juntament amb informació teòrica sobre cada concepte utilitzat. Els *Smart Contracts* desenvolupats han estat dissenyats per tal de gestionar totes les apostes i els *tokens E-bet* de cada compte.

El contracte principal és el *Bet.sol*. En aquest *Smart Contract* hi han totes les funcions que gestionen els estats de les apostes, a més de les funcions necessàries per fer la compra i la venda dels *tokens E-bet.*

Seguidament, s'utilitza l'*Smart Contract ERC20.sol*. En aquest contracte es gestionen tots els balanços de cada compte, es defineix la identificació del *token* i la quantitat del *tokens* que es creen, i es troben totes les funcions respecte a la transferència de *tokens*.

Finalment, la llibreria *SafeMath.sol* proporciona una sèrie de funcions matemàtiques simples i segures per tal d'evitar qualsevol error al sumar, restar, multiplicar o dividir.

5.1. *Bet.sol*

L'anterior *Smart Contract, Bet.sol*, és el principal del projecte. Aquest contracte permet la creació d'una aposta, a més de gestionar-les i realitzar tots els canvis d'estats d'aquestes.

També, apareixen les diferents funcions per comprar i vendre el *tokens E-bet* on es transfereixen aquests *tokens* o *ethers* entre l'usuari que vol realitzar la compra o venda i *l'owner* dels *tokens*.

5.1.1. Declaració de variables

```
using SafeMath for uint256;

enum BetState { open, close, pending }

struct Bet {
    uint256 amount;
    BetState state;
    address bettor1;
    address bettor2;
}

uint256 returnedTokens = 9500;
uint256 numBets;
address payable owner;

mapping(uint256 => Bet) public bets;

uint256 public tokenBuyPrice = 900;
uint256 public tokenSellPrice = 1000;

mapping(address => uint256) tokensToBuy;
mapping(address => uint256) tokensToSell;
```

Fig. 5.1 Variables Bet.sol

La primera línia informa que s'utilitza la llibreria *SafeMath* en totes les variables *uint*. Aquesta llibreria permet que les operacions matemàtiques bàsiques (suma, resta, multiplicació i divisió) s'executin de manera segura. La directiva *using* informa que totes les funcions que contingui *SafeMath* podran ser utilitzades en el moment d'utilitzar qualsevol variable *uint*.

A continuació, es declara la *struct Bet*. L'estructura *Bet* incorpora tota la informació necessària d'una aposta: l'estat de l'aposta, la quantitat de *tokens E-bet* apostats, i les adreces dels dos apostadors. Per tant, podem definir una *struct* com un tipus de dades personalitzat que se li pot establir un nom i unes propietats associades.

Després, podem observar dos variables tipus *unit256*. Aquest tipus de variables són nombres sencers de 256 bits, per tant només poden representar nombres positius.

La variable *returnedToken* s'utilitza per marcar la comissió en tancar una aposta i repartir els premis. Es transfereixen la quantitat proporcional de *E-bet* sobre la següent representació: De 10000 *E-bet*, 9500 s'emporta el guanyador i 500 se'ls queda *l'owner* com a comissió.

La variable *numBets* es tracta d'un simple sumatori de totes les apostes que es creen.

També, podem observar una variable de tipus *address* anomenada *owner*. Aquesta variable es defineix en crear el contracte i estableix que *l'owner* del contracte és l'adreça del creador. Aquest tipus de variables tenen la mateixa mida d'una adreça *d'Ethereum* (20 bytes - 160 bits). A més l'adreça *owner* té l'etiqueta *payable*. Aquest fet indica que l'adreça accepta la rebuda *d'Ethers* d'una altra adreça.

La següent variable que podem observar, es tracta d'un *mapping* de *uint256* a l'estructura *Bet*. En ell, es referèncien totes les apostes amb un número que s'utilitza com identificador de l'aposta. Per tant, els *mappings* s'utilitzen per estructurar altres tipus de dades, com *booleans*, *uints*, *address* i *structs*.

A continuació, estan definits els valors per realitzar la compra-venda de *E-bet*. Per comprar 900 *E-bet* es necessita 1 *Ether*, en canvi, per rebre 1 *Ether* es necessiten 1000 *E-bet*.

Finalment, estan definits uns *mappings* per poder realitzar correctament les transaccions de compra-venda de *tokens*. En aquestes variables es guarden les quantitats de *tokens* que una adreça està pendent de comprar o de vendre.

5.1.2. *Events*

```
event BetPending(uint256 timestamp, uint256 id);
event BetOpened(uint256 timestamp, uint256 id);
event BetClosed(uint256 timestamp, uint256 id);

event BuyTokensSendEthers(address buyer, uint256 tokens);
event BuyTokensSendTokens(address buyer, uint256 tokens);
event SellTokensSendEthers(address seller, uint256 tokens);
event SellTokensSendTokens(address seller, uint256 tokens);
```

Fig. 5.2 Events Bet.sol

Els *events* són la forma en què es pot retorna qualsevol informació al llençar una funció d'un *Smart Contract*. A més, es guarden en una estructura especial dins la transacció anomenada registre de la transacció. En emmagatzemar-se en la transacció es podrà accedir a la seva informació sempre que el bloc es confirmi i la transacció sigui correcta.

Com es pot observar anteriorment, existeixen 3 *events* relacionats amb els canvis d'estat de les apostes, i 4 *events* relacionats amb la compra/venda dels *tokens E-bet*.

- *BetPending*: *L'event BetPending* s'emet en el moment en què una aposta s'estableix en estat *Pending*, i retorna el moment en què es llença la funció i l'id de l'aposta.
- *BetOpened*: L'*event BetOpened* s'emet en el moment en què una aposta s'estableix en estat Open, i retorna el moment en què es llença la funció i l'id de l'aposta.
- *BetClosed*: L*'event BetClosed* s'emet en el moment en què una aposta s'estableix en estat *Close*, i retorna el moment en què es llença la funció i l'id de l'aposta.
- *BuyTokensSendEthers*: *L'event BuyTokensSendEthers* s'emet en el moment en què un usuari crida la funció *buyTokensPassEthers* on envia els *Ethers* que vol intercanviar pels *tokens E-bet* a l'*owner* dels *tokens*.
- *BuyTokensSendTokens*: *L'event BuyTokensSendTokens* s'emet en el moment en què l'owner crida la funció *buyTokensPassTokens* on envia els *tokens E-bet* a l'usuari que ha pagat *Ethers* per a ells.
- SellTokensSendTokens: *L'event SellTokensSendTokens* s'emet en el moment en què un usuari crida la funció *sellTokensPassTokens* on envia els tokens E-bet que vol intercanviar per *Ethers* a l'*owner* dels *tokens*.
- SellTokensSendEthers: *L'event SellTokensSendTokens* s'emet en el moment en què *l'owner* crida la funció *sellTokensPassTokens* on envia els *Ethers* a l'usuari que ha pagat *tokens E-bet* per a ells.

5.1.3. Constructor

```
constructor () public {
    owner = msg.sender;
}
```

Fig. 5.3 Constructor Bet.sol

Un constructor és una funció única (només es permet crear un constructor) i opcional que s'executa en el moment en què es crea i s'introdueix el contracte dins la *Blockchain*.

En el constructor del *Smart Contract* simplement s'estableix que l'adreça *owner* del contracte és l'adreça que ha realitzat la creació d'aquest.

5.1.4. *Modifiers*

```
modifier onlyOwner {
    require(msg.sender == owner);
    _;
}
```

Fig. 5.4 Modifiers Bet.sol

Els *modifiers* s'utilitzen per comprovar que la condició establerta en el *modifier* es compleix abans de realitzar la funció cridada. A més, és una forma de netejar i evitar la redundància de codi.

El *modifier onlyOwner* estableix que la funció a la qual s'introdueix aquest *modifier* només la pot cridar *l'owner* del *Smart Contract*.

5.1.5. Funcions

5.1.5.1. BetCreate

```
function betCreate(uint256 _amount) external {
    require(balances[msg.sender] >= _amount);
    require(_amount > 0);
    approve(msg.sender, _amount);
    transfer(owner, _amount);
    bets[numBets++] = Bet(_amount, BetState.pending, msg.sender, address(0));
    emit BetPending(block.timestamp, numBets);

}
```

Fig. 5.5 Funció betCreate() de Bet.sol

La funció *betCreate* és cridada per un usuari en el moment en què vol crear una aposta i ha d'enviar com a paràmetre la quantitat de *tokens E-bet* que vol apostar.

A l'inici de la funció es comprova que la quantitat d'*E-bet* apostats sigui inferior que la quantitat d'*E-bet* què disposa l'adreça de l'usuari apostant. També, es

comprova que aquesta quantitat apostada sigui superior a 0. En cas que no es compleixi una d'aquestes condicions, la transacció es reverteix.

Posteriorment, s'aprova i es transfereix a *l'owner* els *tokens* indicats. A més, es crea l'aposta indicant la quantitat d'*E-bet*, l'estat s'estableix a *Pending*, el *bettor1* és l'usuari que ha cridat l'aposta, i en el valor del *bettor2* introduïm l'adreça 0x00.

Finalment, s'emet l'*event BetPending* adjuntant el *timestamp* i l'identificador de l'aposta.

5.1.5.2. BetOpen

```
function betOpen(uint256 _amount, uint256 _id) external {
    require(bets[_id].state == BetState.pending);
    require(balances[msg.sender] >= _amount);
    approve(msg.sender, _amount);
    transfer(owner, _amount);
    bets[_id].bettor2 = msg.sender;
    bets[_id].state = BetState.open;
    emit BetOpened(block.timestamp, _id);
}
```

Fig. 5.6 Funció betOpen() de Bet.sol

La funció *betOpen* és cridada per un usuari en el moment en què vol acceptar una aposta que ha creat un altre usuari. A més, ha d'enviar com a paràmetre la quantitat de *tokens E-bet* i l'identificador de l'aposta que es vol acceptar.

A l'inici de la funció es comprova que l'estat de l'aposta sigui *Pending* i que la quantitat d'*E-bet* apostats sigui inferior que la quantitat d'*E-bet* què disposa l'adreça de l'usuari apostant. També, es comprova que aquesta quantitat apostada sigui superior a 0. En cas que no es compleixi una d'aquestes condicions, la transacció es reverteix.

Posteriorment, s'aprova i es transfereix a *l'owner* els *tokens* indicats. A més, s'introdueix l'adreça de l'usuari que ha cridat l'aposta en el paràmetre *bettor2* i s'estableix l'aposta en estat *Open*.
Finalment, s'emet l'*event BetOpened* adjuntant el *timestamp* i l'identificador de l'aposta.

5.1.5.3. BetClose

```
function betClose(address _winner, uint256 _amount, uint256 _id) external onlyOwner {
    require(bets[_id].state == BetState.open);
    approve(owner, _amount.mul(2));
    transferFrom(owner, _winner, _amount.mul(2).mul(returnedTokens).div(10000));
    bets[_id].state = BetState.close;
    emit BetClosed(block.timestamp, _id);
}
```

Fig. 5.7 Funció betClose() de Bet.sol

La funció *betClose* és cridada per *l'owner* en el moment en què es vol tancar una aposta. Aquesta funció es llença quan la partida acaba i s'obté el guanyador de l'aposta. *L'owner* ha d'enviar com a paràmetre el guanyador de l'aposta, la quantitat d'*E-bet* apostats i la identificació de l'aposta.

A l'inici de la funció es comprova que l'estat de l'aposta sigui *Open*. En cas que no es compleixi aquesta condició, la transacció es reverteix.

Posteriorment, s'aprova i es transfereix al guanyador la quantitat apostada pels dos usuaris restant-li una petita comissió que es queda *l'owner*. A més, s'estableix l'aposta en estat *Close*.

Finalment, s'emet l'*event BetClosed* adjuntant el *timestamp* i l'identificador de l'aposta.

5.1.5.4. BetCloseFromPending

```
function betCloseFromPending(address _bettor1, uint256 _amount, uint256 _id) external onlyOwner {
    require(bets[_id].state == BetState.pending);
    approve(owner, _amount);
    transferFrom(owner, _bettor1, _amount.mul(returnedTokens).div(10000));
    bets[_id].state = BetState.close;
    emit BetClosed(block.timestamp, _id);
}
```

Fig. 5.8 Funció betCloseFromPending() de Bet.sol

La funció *betCloseFromPending* és cridada per *l'owner* en el moment en què es vol tancar una aposta. Aquesta funció es llença quan ningú ha acceptat l'aposta. *L'owner* ha d'enviar com a paràmetre l'adreça del creador de l'aposta, la quantitat d'*E-bet* apostats i la identificació de l'aposta.

A l'inici de la funció es comprova que l'estat de l'aposta sigui *Pending*. En cas que no es compleixi aquesta condició, la transacció es reverteix.

Posteriorment, s'aprova i es transfereix al creador de l'aposta la quantitat apostada restant-li una petita comissió que es queda *l'owner*. A més, s'estableix l'aposta en estat *Close*.

Finalment, s'emet l'*event BetClosed* adjuntant el *timestamp* i l'identificador de l'aposta.

5.1.5.5. BetState

```
function betState(uint256 _id) external view returns (BetState state) {
    state = bets[_id].state;
}
```

Fig. 5.9 Funció betState() de Bet.sol

La funció betState retorna l'estat en què es troba l'aposta la qual s'envia el seu identificador com a paràmetre.

5.1.5.6. BuyTokensPassEthers

```
function buyTokensPassEthers() external payable {
    require(msg.value > 0);
    tokensToBuy[msg.sender] = tokensToBuy[msg.sender].add(msg.value.mul(tokenBuyPrice).div(10**18));
    emit BuyTokensSendEthers(msg.sender, msg.value.mul(tokenBuyPrice).div(10**18));
    owner.transfer(msg.value);
}
```

Fig. 5.10 Funció buyTokensPassEthers() de Bet.sol

La funció *buyTokensPassEthers* és cridada per un usuari en el moment en què vol comprar uns *E-bet*.

A l'inici de la funció es comprova que la quantitat d'*E-bet* que es vol comprar sigui superior a 0. En cas que no es compleixi aquesta condició, la transacció es reverteix.

Posteriorment, s'agafa la quantitat de *Wei's* (*msg.value*) introduïts en la transacció, es realitza la conversió a *E-bet,* i s'afegeixen en el *mapping* que guarda la quantitat de *tokens* que un usuari vol comprar i que encara no ha rebut.

Després, s'emet l'*event BuyTokensSendEthers* adjuntant l'adreça de l'usuari que vol comprar els *tokens E-bet* i la quantitat d'*E-bet* que ha de rebre.

Finalment, es transfereix la quantitat de *Wei*'s introduïts en la transacció a *l'owner*.

5.1.5.7. BuyTokensPassTokens

```
function buyTokensPassTokens(address _address, uint _tokens) external onlyOwner {
    require(_tokens > 0);
    require(tokensToBuy[_address] >= _tokens);
    tokensToBuy[_address] = tokensToBuy[_address].sub(_tokens);
    approve(_address, _tokens);
    transfer(_address, _tokens);
    emit BuyTokensSendTokens(msg.sender, _tokens);
}
```

Fig. 5.11 Funció buyTokensPassTokens() de Bet.sol

La funció *buyTokensPassTokens* és cridada per l'owner en el moment en què rep la confirmació que un usuari li ha enviat els *Ethers* per comprar *tokens E-bet*. *L'owner* ha d'enviar com a paràmetre l'adreça de l'usuari que vol comprar els *E-bet* i la quantitat d'*E-bet* que es volen comprar.
A l'inici de la funció es comprova que la quantitat d'*E-bet* que es vol comprar sigui superior a 0. També, es comprova que la quantitat d'*E-bet* que vol comprar l'usuari sigui igual o inferior a la quantitat de *tokens E-bet* que encara no ha rebut. En cas que no es compleixi una d'aquestes condicions, la transacció es reverteix.

Posteriorment, es resta en el *mapping* que guarda la quantitat de *tokens* que un usuari vol comprar i que encara no ha rebut la quantitat que s'envia a l'usuari. Després, s'aprova i es transfereix a l'usuari els *tokens E-bet* indicats.

Finalment, s'emet l'*event BuyTokensSendTokens* adjuntant l'adreça de *l'owner* i la quantitat d'*E-bet* que ha de rebre.

5.1.5.8. SellTokensPassTokens

```
function sellTokensPassTokens(uint _tokens) external {
    require(_tokens > 0);
    tokensToSell[msg.sender] = tokensToSell[msg.sender].add(_tokens);
    approve(owner, _tokens);
    transfer(owner, _tokens);
    emit SellTokensSendTokens(msg.sender, _tokens);
}
```

Fig. 5.12 Funció sellTokensPassTokens() de Bet.sol

La funció *sellTokensPassTokens* és cridada per un usuari en el moment en què vol vendre uns *E-bet.* L'usuari ha d'enviar com a paràmetre la quantitat d'E-bet que vol vendre.

A l'inici de la funció es comprova que la quantitat d'*E-bet* que es vol vendre sigui superior a 0. En cas que no es compleixi aquesta condició, la transacció es reverteix.

Posteriorment, s'agafa la quantitat d'*E-bet* i s'afegeixen en el *mapping* que guarda la quantitat de *tokens* que un usuari vol vendre i que encara no ha rebut.

Després, s'aprova i es transfereix a *l'owner* els *tokens E-bet* indicats.

Finalment, s'emet l'*event SellTokensSendTokens* adjuntant l'adreça de l'usuari que vol vendre els t*okens E-bet* i la quantitat d'*E-bet* què vol vendre.

5.1.5.9. SellTokensPassEthers

```
function sellTokensPassEthers(address payable _address, uint _tokens) external payable onlyOwner {
    require(msg.value > 0);
    require(tokensToSell[_address] >= _tokens);
    tokensToSell[_address] = tokensToSell[_address].sub(_tokens);
    emit SellTokensSendEthers(msg.sender, _tokens);
    _address.transfer(msg.value);
}
```

Fig. 5.13 Funció sellTokensPassEthers() de Bet.sol

La funció *sellTokensPassEthers* és cridada per l'owner en el moment en què rep la confirmació que un usuari li ha enviat els *tokens E-bet* per vendre'ls.

L'owner ha d'enviar com a paràmetre l'adreça de l'usuari que vol vendre els *E-bet* i la quantitat d'*E-bet* que es volen vendre.

A l'inici de la funció es comprova que la quantitat de *Wei's* introduïts en la transacció sigui superior a 0. També, es comprova que la quantitat d'*E-bet* que vol vendre l'usuari sigui igual o inferior a la quantitat de *tokens E-bet* que encara no ha rebut. En cas que no es compleixi una d'aquestes condicions, la transacció es reverteix.

Posteriorment, es resta en el *mapping* que guarda la quantitat de *tokens*, que un usuari vol vendre i que encara no ha rebut, la quantitat que s'envia a l'owner.

Després, s'emet l'*event SellTokensSendEthers* adjuntant l'adreça de *l'owner* i la quantitat d'*E-bet* que ha rebut.

Finalment, es transfereix la quantitat de *Wei's* introduïts en la transacció a l'usuari.

5.2. *ERC20.sol*

L'Smart Contract ERC20.sol és l'estàndard que es basa a construir una sèrie de normes per tal de definir la gestió de balanços de cada adreça i l'administració de les transferències de *tokens*.

Les primeres funcions són funcions de metadades les quals serveixen per donar una identificació als *tokens* creats.

A continuació, hi ha una sèrie de funcions de tipus *view*. Aquestes funcions no modifiquen l'estat de cap variable, només serveixen per consultar el paràmetre que es retorna a la funció que es cridi.

Finalment, hi ha les funcions per poder realitzar o poder aprovar les transferències de *tokens E-bet* d'una adreça a una altra.

5.2.1. Definició de variables

```
using SafeMath for uint256;

uint256 numTokens;

// Tokens data and metadata arrays
mapping(address => uint256) balances;
mapping(address => mapping(address => uint256)) allowed;
```

Fig. 5.14 Definició de variables de ERC20.sol

La primera línia informa que s'utilitza la llibreria *SafeMath* en totes les variables *uint*. Aquesta llibreria permet que les operacions matemàtiques bàsiques (suma, resta, multiplicació i divisió) s'executin de manera segura.

A continuació, es defineix la quantitat de *tokens* que es crearan en una variable uint256 anomenada *numTokens*.

Finalment, estan definits el mapping que defineix la quantitat de token E-bet que té cada adreça, i el *mapping* que defineix la quantitat de tokens E-bet que una adreça té aprovats per enviar a una altra adreça.

5.2.2. Constructor

```
constructor() public {
    numTokens = 1000000 * 10**uint(18);
    balances[msg.sender] = numTokens;
    emit Transfer(address(0), address(this), numTokens);
}
```

Fig. 5.15 Constructor de ERC20.sol

En el constructor del *Smart Contract ERC20.sol* s'estableix el número de tokens que es creen i s'envien a l'usuari que crea el contracte. Finalment, s'emet *l'event Transfer* que retorna l'adreça 0x00, l'adreça del compte que ha creat el contracte i el número de *tokens* creats.

5.2.3. Funcions de metadades

```
function name() external pure returns (string memory _name){
    return "E-bets";
}

function symbol() external pure returns (string memory _symbol){
    return "⚖";
}
```

Fig. 5.16 Funcions de metadades de ERC20.sol

Aquestes funcions de metadades serveixen per identificar el *token* creat.
La funció ***name*** retorna el nom del *tokens*, anomenat *E-bet.* A més, la funció *symbol*() retorna el símbol identificatiu del *token*.

5.2.4. Funcions

```
function balanceOf(address _owner) public view returns (uint256 _balance) {
    return balances[_owner];
}
```

Fig. 5.17 Funció balanceOf() de ERC20.sol

La funció ***balanceOf*** retorna la quantitat de *tokens E-bet* que té l'adreça _owner pasada com a paràmetre.

Paràmetres:

- *address _owner*: Adreça del compte el qual es vol saber la quantitat de *tokens E-bet.*

Return:

- *uint256 _balance*: Número de *tokens E-bet.*

```
function allowance(address _tokenOwner, address _spender) public view returns (uint remaining) {
    return allowed[_tokenOwner][_spender];
}
```

Fig. 5.18 Funció allowance() de ERC20.sol

La funció ***allowance*** retorna el nombre de *tokens* que es pot retirar d'una adreça donada com a paràmetre.

Paràmetres:

- *address _tokenOwner*: Adreça del compte de l'owner dels tokens E-bet.
- *address _spender*: Adreça del compte el qual gasta els tokens E-bet.

Return:

- *uint256 remaining*: Número de tokens E-bet que encara pot gastar el _spender.

```
function transfer(address _to, uint256 _tokens) public returns (bool success) {
    balances[msg.sender] = balances[msg.sender].sub(_tokens);
    balances[_to] = balances[_to].add(_tokens);
    emit Transfer(msg.sender, _to, _tokens);
    return true;
}
```

Fig. 5.19 Funció transfer() de ERC20.sol

Amb la funció **transfer** es transfereixen tokens, des de la pròpia compta a una adreça donada com a paràmetre.

Paràmetres:

- *address _to*: Adreça del compte al qual es volen transferir els tokens E-bet.
- *uint256 _tokens*: Número de *tokens E-bet* que es transfereixen a l'adreça _to.

Return:

- *bool succes*: Retorna true si s'ha realitzat correctament la transferència de *tokens E-bet.*

```
function approve(address _spender, uint256 _tokens) public returns (bool success) {
    allowed[msg.sender][_spender] = _tokens;
    emit Approval(msg.sender, _spender, _tokens);
    return true;
}
```

Fig. 5.20 *Funció approve() de ERC20.sol*

Amb la funció ***approve*** es permet la retirada de *tokens* d'una adreça donada com a paràmetre.

Paràmetres:

- *address _spender:* Adreça del compte el qual es permet transferir els tokens *E-bet.*
- *uint256 _tokens*: Número de *tokens E-bet* que pot transferir l'adreça _spender.

Return:

- *bool succes*: Retorna true si s'ha realitzat correctament la transferencia de tokens *E-bet.*

```
function transferFrom(address _from, address _to, uint256 _tokens) public returns (bool success) {
    balances[_from] = balances[_from].sub(_tokens);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_tokens);
    balances[_to] = balances[_to].add(_tokens);
    emit Transfer(_from, _to, _tokens);
    return true;
}
```

Fig. 5.21 *Funció transferFrom() de ERC20.sol*

Amb la funció ***transferFrom*** es transfereixen *tok*ens des d'una adreça donada com a paràmetre a una altra adreça també donada com a paràmetre.

Paràmetres:

- *address _from*: Adreça del compte des del qual es volen transferir els *tokens E-bet.*
- *address _to*: Adreça del compte al qual es volen transferir els *tokens E-bet.*
- uint256 _tokens: Número de *tokens E-bet* que es transfereixen des de l'adreça _from a l'adreça _to.

Return:

- *bool succes*: Retorna *true* si s'ha realitzat correctament la transferència de *tokens E-bet.*

5.3. *IERC20.sol*

```
pragma solidity >= 0.4.0 <0.6.0;

interface IERC20 {

    // METADATA FUNCTIONS
    function name() external pure returns (string memory _name);
    function symbol() external pure returns (string memory _symbol);

    function totalSupply() external view returns (uint);
    function balanceOf(address tokenOwner) external view returns (uint balance);
    function allowance(address tokenOwner, address spender) external view returns (uint remaining);
    function transfer(address to, uint tokens) external returns (bool success);
    function approve(address spender, uint tokens) external returns (bool success);
    function transferFrom(address from, address to, uint tokens) external returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

Fig. 5.22 Interfície ERC20.sol

Una interfície és una definició d'una sèrie de funcions que existiran en el contracte el qual s'implementarà. A més, existeixen una sèrie de restriccions que s'han de complir a l'hora de desenvolupar una interfície:

- No es pot implementar cap funció.
- No pot heretar ni altres interfícies ni contractes.
- No pot declarar ni variables d'estat ni el constructor.
- Totes les funcions que es declarin s'han d'establir com a *external*.

La interfície *IERC20.sol (**Fig. 5.22**)* defineix tant les funcions com els *events* que utilitza el contracte *ERC20.sol* explicat anteriorment.

5.4. *SafeMath.sol*

La llibreria *SafeMath.sol* proporciona la possibilitat de realitzar les operacions aritmètiques de *Solidity* amb una comparació afegida.

Amb la introducció d'un *require* en cada funció evita desbordaments al realitzar qualsevol de les funcions matemàtiques anteriors.

Finalment, si es produeix algun desbordament, es reverteix la transacció.

5.4.1. Funcions

```
function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
    c = a + b;
    require(c >= a);
}
```

***Fig. 5.23** Funció add() de SafeMath.sol*

La funció ***add*** retorna la suma de dos valors passats com a paràmetre.

Paràmetres: *uint256 a i uint256 b*: Primer i segon número per realitzar la suma.

Return: *uint256 c*: Resultat de la suma entre a i b.

```
function sub(uint256 a, uint256 b) internal pure returns (uint256 c) {
    require(b <= a);
    c = a - b;
}
```

Fig. 5.24 *Funció sub() de SafeMath.sol*

La funció ***sub*** retorna la resta de dos valors passats com a paràmetre.

Paràmetres: *uint256 a i uint256 b*: Primer i segon número per realitzar la resta.

Return: *uint256 c*: Resultat de la resta entre a i b.

```
function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
    c = a * b;
    require(a == 0 || c / a == b);
}
```

Fig. 5.25 *Funció mul() de SafeMath.sol*

La funció ***mul*** retorna la multiplicació de dos valors passats com a paràmetre.

Paràmetres: *uint256 a i uint256 b*: Primer i segon número per realitzar la multiplicació.

Return: *uint256 c*: Resultat de la multiplicació entre a i b.

```
function div(uint256 a, uint256 b) internal pure returns (uint256 c) {
    require(b > 0);
    c = a / b;
}
```

Fig. 5.26 *Funció div() de SafeMath.sol*

La funció ***div*** retorna la divisió de dos valors passats com a paràmetre.

Paràmetres: *uint256 a i uint256 b*: Primer i segon número per realitzar la divisió.

Return: *uint256 c*: Resultat de la divisió entre a i b.

CAPÍTOL 6. *BACK-END*

En aquest capítol s'explica tota la informació que envolta al desenvolupament del *Back-End* del projecte.

Inicialment, s'explica la base de dades utilitzada. A continuació, s'explica *l'API* del projecte separada segons cada arxiu de rutes desenvolupat, la configuració per utilitzar *l'API* de *Riot Games*, el *Cron* i els *Sockets* utilitzats. Finalment, s'expliquen els mecanismes de seguretat utilitzats.

6.1. Base de dades

La base de dades utilitzada en aquest projecte es tracta d'una base de dades *MongoDB*. Aquesta base de dades és del tipus *NOSQL* i utilitza documents en un format semblant a *JSON*.

Per connectar el nostre *Back-End* amb la base de dades s'ha utilitzat el paquet anomenat *Mongoose*. A continuació, per poder declarar i posteriorment guardar dades en la base de dades s'ha creat un arxiu per cada Col·lecció de *Mongo* on hi ha declarat un *Schema*. En aquests esquemes hi ha introduïts tots els Camps que pot arribar a tenir un Document d'aquesta Col·lecció.

6.1.1. *Mongoose*

Fig. 6.1 Icona de la llibreria Mongoose

Mongoose és una llibreria encarregada de crear objectes models per la introducció de dades en una base de dades *MongoDB*. En la realització d'aquests objectes models, *Mongoose* inclou diferents funcions de validacions i requeriments per seguir l'estructura que es requereix donant sempre l'elasticitat que aporta la flexibilitat dels Documents de *MongoDB*.

Aquesta llibreria inclou una forma molt senzilla per realitzar la connexió amb una base de dades *MongoDB*. Amb la següent comanda es realitza una connexió en local a la base de dades *MongoDB* anomenada *Myapp*.

```
mongoose.connect("mongodb://localhost:27017/tfg",{
    useNewUrlParser: true,
    reconnectTries : Number.MAX_VALUE,
    autoReconnect : true,
    reconnectInterval: 500
});
```

Fig. 6.2 Connexió amb Mongo DB

En aquest projecte, i per mantenir un control sobre l'estat de la base de dades, s'ha utilitzat un *events*. Aquests *events* es llencen en el moment en què es connecta, hi ha un error, o es deconnecta la base de dades.

```
mongoose.connect("mongodb://localhost:27017/tfg",{
    useNewUrlParser: true,
    reconnectTries : Number.MAX_VALUE,
    autoReconnect : true,
    reconnectInterval: 500
});

// When successfully connected
mongoose.connection.on('connected', () => {
    console.log('dbevent: open');
});

// When successfully reconnected
mongoose.connection.on('reconnected', () => {
    console.log('dbevent: reconnected');
});

// If the connection throws an error
mongoose.connection.on('error', (err) => {
    console.log(`dbevent: error: ${err}`);
});

// When the connection is disconnected
mongoose.connection.on('disconnected', () => {
    console.log('dbevent: disconnected');
    mongoose.connect("mongodb://localhost:27017/tfg",{
        useNewUrlParser: true,
        reconnectTries : Number.MAX_VALUE,
        autoReconnect : true,
        reconnectInterval: 500
    });
```

Fig. 6.3 Funcions connexió, reconnexió, error i desconnecció de MongoDB

Com es pot observar en el codi anterior (***Fig. 6.3)*** que es troba en l'arxiu principal del *Back-End* (*app.js*), s'intenta realitzar una connexió a la base de dades "*tfg*". A més, si no s'aconsegueix, s'intenta tornar a connectar cada 500ms, fins a aconseguir-ho.

A continuació, s'observen els *events connected, reconnected* i *error* que informen si s'ha produït en algun moment un canvi en l'estat de la base de dades.

I, finalment, l'*event disconnected* informa que la base de dades s'ha desconnectat, i intenta tornar-se a connectar executant el mètode *connect*.

Una vegada s'ha establert la connexió, s'ha de gestionar les dades a guardar. Per poder emmagatzemar dades en una Col·lecció, *Mongoose* necessita que es creïn uns esquemes. Un *Schema* és un mapeig que es realitza amb una Col·lecció de *MongoDB* on es defineixen els Camps que pot incorporar un Document d'aquesta Col·lecció.

Cada camp ha d'estar definit per un *SchemaType*. Els *SchemaType* són els tipus de dades al qual un camp pot estar associat. Existeixen els següents *SchemaTypes*:

String, Number, Date, Buffer, Boolean, Mixed, ObjectId, Array, Decimal128, Map.

A més, dins de cada Camp es poden definir diferents opcions com el valor per defecte que té el Camp determinat o si es requereix introduir aquest Camp per poder guardar el Document.

Una vegada s'ha creat el *Schema*, s'ha de realitzar una conversió d'aquest *Schema* a un Model. Al realitzar aquesta conversió, s'ha d'especificar el nom que ha de tenir la Col·lecció a la qual es vol guardar dades.

6.1.2. Models

Els models que s'han creat per aquest projecte són els següents:

- *Bet*: Aquest model defineix els camps que pot tenir un Document *Bet*. En ell, es defineix la informació sobre els *tokens* apostats, la duració i els apostadors. A més, es referència a *l'id* del joc al qual es realitza l'aposta amb el Camp *GameId*. Finalment, es marca el guanyador de l'aposta en el Camp *winner*.
- *Game*: Aquest model defineix els camps que pot tenir un Document *Game*. En ell, es defineix la informació sobre la partida, tant el moment d'inici, com el tipus de partida, la regió i un *Array* amb la informació de tots els participants. Finalment, es marca el guanyador de la partida en el Camp *winner*.
- *Summoner*: Aquest model defineix els camps que pot tenir un Document *Summoner*. En ell, es guarda el nom del *Summoner*, el nivell i l'id encriptat per realitzar consultes a l'*API* de *Riot Games* de forma més ràpida i senzilla, i així estalviar-nos una petició en cada moment que es vulgui realitzar una petició de recerca d'informació del *Summoner*.
- *User*: Aquest model defineix els camps que pot tenir un Document *User*. En ell, es defineix el nom d'usuari i un camp anomenat *password* on s'introdueix el *hash* de la contrasenya. A més, es guarda una imatge de perfil, un *Array* amb els amics i els *Summoners* preferits de l'usuari així com les estadístiques de les apostes. Finalment, s'emmagatzema la informació necessària per realitzar les funcions dels *JSON Web Tokens* de forma correcta.
- *Chat*: Aquest model defineix els camps que pot tenir un Document *Chat*. En ell, es defineix el nom de la sala, els usuaris que es troben dins, el moment en què s'ha creat la sala, un *Array* de missatges i informació sobre l'últim missatge enviat.
- Message: Aquest model defineix els camps que pot tenir un Document *Message*. En ell, es defineix el missatge, el nom de la sala on s'ha produït el missatge, l'emissor i receptor, el moment en què s'ha creat i si ha estat visualitzat o no pel receptor.

En l'*Annex D: Models de dades a guardar a MongoDB* es troben els models de la base de dades creats.

6.2. API

Per poder connectar el *Back-End* amb el *Front-End,* s'ha creat una *API* on s'exposen totes les rutes diferenciades segons la matèria de la petició. Aquesta *API* ha estat creada amb el *framework* anteriorment explicat, anomenat *Express*.

6.2.1. *Bets API*

En aquesta *API* es mostren totes les peticions que es poden realitzar per obtenir la informació necessària respecte a les apostes i els *tokens*.

```
router.post('/create', md_auth.ensureAuth, betScripts.createBet);
router.post('/accept', md_auth.ensureAuth, betScripts.acceptBet);
router.post('/closeFromPending', md_auth.ensureAuth, betScripts.closeFromPending);
router.post('/search', md_auth.ensureAuth, betScripts.searchBet);
router.post('/pendingBets', md_auth.ensureAuth, betScripts.getPendingBets);
router.post('/getBet', md_auth.ensureAuth, betScripts.getBet);
router.post('/getBetsFromUser', md_auth.ensureAuth, betScripts.getBetsFromUser);
router.post('/getBetsPendingFromUser', md_auth.ensureAuth, betScripts.getBetsPendingFromUser);
router.post('/getBetsOpenFromUser', md_auth.ensureAuth, betScripts.getBetsOpenFromUser);

router.post('/tokens/transfer', md_auth.ensureAuth, betScripts.transferTokens);
router.post('/tokens/getFromAddress', md_auth.ensureAuth, betScripts.getTokensFromAddress);
```

Fig. 6.4 API on hi ha les peticions de Bets

Com es pot observar en la ***Fig. 6.4*** podem separar les peticions segons si modifiquen informació d'apostes o dels *tokens*. Les peticions de les apostes s'utilitzen essencialment per modificar alguna informació d'una aposta guardada en la base de dades, o per retornar una o grups d'apostes.

Les peticions dels *tokens* s'utilitzen per llançar una petició de transferència de *tokens* per part de *l'owner* del contracte (si es compleixen unes condicions específiques) o per retornar el nombre de *tokens E-bet* d'una adreça.

6.2.2. *User API*

```
router.post('/login', userScripts.login);
router.post('/register', userScripts.register);

router.post('/getUserFromId', md_auth.ensureAuth, userScripts.getUserFromId);
router.post('/getUserFromUsername', md_auth.ensureAuth, userScripts.getUserFromUsername);
router.post('/getNumberTokens', md_auth.ensureAuth, userScripts.getNumberTokens);
router.post('/editUser', md_auth.ensureAuth, userScripts.editUser);
router.post('/editFavouriteSummoner', md_auth.ensureAuth, userScripts.editFavouriteSummoner);

router.post('/search', md_auth.ensureAuth, userScripts.searchUser);

router.post('/friends/add', md_auth.ensureAuth, userScripts.addFriend);
router.post('/friends/delete', md_auth.ensureAuth, userScripts.deleteFriend);

router.post('/buyTokensPassTokens', md_auth.ensureAuth, userScripts.buyTokensPassTokens);
router.post('/sellTokensPassEthers', md_auth.ensureAuth, userScripts.sellTokensPassEthers);
```

Fig. 6.5 API on hi ha les peticions de User

Com es pot observar en la ***Fig. 6.5,*** podem separar les peticions segons si realitzem funcions d'autenticació, d'obtenció o edició d'informació d'usuari, o compra-venda de *tokens E-bet*.

Les dues primeres peticions s'utilitzen en el moment de registrar o d'entrada d'un usuari al seu compte. Són les úniques funcions que no necessiten un *token* d'autenticació per poder executar-se.

Les següents peticions es criden per obtenir informació de l'usuari o per editar algun aspecte d'informació personal o el *Summoner* preferit d'un usuari.
Finalment, les dues últimes peticions es criden per completar el procés de compra-venda dels *tokens E-bet*.

6.2.3. *Summoner API*

```
router.post('/searchSummoner', md_auth.ensureAuth, summonerScripts.searchSummonerBet);
router.post('/searchSummonerInfo', md_auth.ensureAuth, summonerScripts.searchSummonerInfo);
```

Fig. 6.6 API on hi ha les peticions de Summoner

Com es pot observar en la ***Fig. 6.6,*** les dues peticions de *l'API* de *Summoner* serveixen per obtenir informació del *Summoner* i informació de la partida que està disputant en directe el *Summoner* escollit.

6.2.4. *Chat API*

```
router.post('/getRoom', md_auth.ensureAuth, chatScript.getChatRoom);
router.post('/getRoomById', md_auth.ensureAuth, chatScript.getChatRoomById);
router.post('/getMessages', md_auth.ensureAuth, chatScript.getMessages);
router.post('/messagesNotSeen', md_auth.ensureAuth, chatScript.getMessagesNotSeen);
router.post('/lastView', md_auth.ensureAuth, chatScript.lastView);
router.post('/getChats', md_auth.ensureAuth, chatScript.getChats);
router.post('/newChat', md_auth.ensureAuth, chatScript.createChat);
```

Fig. 6.7 API on hi ha les peticions de Xat

Com es pot observar en la ***Fig. 6.7,*** es mostren totes les peticions que es necessiten per obtenir tota la informació necessària dels xats, ja sigui una conversa, l'última visita o la llista de converses actives d'un usuari.

6.3. *Riot Games API*

Per poder obtenir tota la informació de les partides de *League of Legends* i la informació de tots els *Summoners* s'ha d'utilitzar *l'API* oficial de *Riot Games*.

6.3.1. Registre en *l'API* de *Riot Games*

Per utilitzar *l'API* de *Riot Games* s'ha de tenir un compte de *Riot Games* i entrar a la mateixa sessió d'aquesta web:

https://developer.riotgames.com/

Una vegada ens trobem dins, hem d'obtenir una *API Key*. Cada petició a l'*API* de *Riot Games* requereix una clau *d'API*. S'ha d'incloure el paràmetre *api_key* en cada petició.

Les peticions es realitzen atacant a la següent URL:

https://euw1.api.riotgames.com/lol

A més, s'ha d'incloure les següents capçaleres (***Fig. 6.8)*** en les opcions de cada petició.

```
let optionsRequest = {
    url: '',
    headers: {
        "Origin": "https://developer.riotgames.com",
        "Accept-Charset": "application/x-www-form-urlencoded; charset=UTF-8",
        "X-Riot-Token": "RGAPI-d9e505c4-5fe7-4e40-ba04-d55d92496c64",
        "Accept-Language": "es-ES,es;q=0.9",
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
            "Chrome/72.0.3626.119 Safari/537.36"
    }
};
```

Fig. 6.8 Capçaleres que s'han d'incloure en realitzar sol·licituds a l'API de Riot Games

6.3.2. Normativa Riot Games

Riot Games marca una estructura i una normativa que s'ha de seguir de manera estricta. A l'hora de desenvolupar un projecte, s'ha de seguir les normatives explicades en l'*Annex C: Normativa Riot Games.*

6.4. *Cron*

Per poder mantenir un control dels estats de les partides i de les apostes s'executa un *Cron* cada 1 minut.

Un *Cron* és un executador de funcions en segon pla que llença les funcions introduïdes en aquest Cron en intervals marcats.

En aquest projecte s'utilitza el mètode *CronJob* de la llibreria *cron*. Amb aquest mètode s'utilitza 3 funcions on es comproven les apostes en estat *Pending*, *Open* i *Close*.

6.4.1. Funcions

- Comprovació apostes en estat *Pending*

En aquesta funció es comprova que les apostes en estat *Pending* no hagin finalitzat. Per realitzar aquesta comprovació, s'assegura que el camp *duration* de l'objecte *bet* sigui major al moment actual. En cas negatiu, es realitza una transacció des del compte de *l'owner* realitzant el mètode *betCloseFromPending* de l'*Smart Contract* (tanca l'aposta) i s'actualitza la base de dades canviant el camp *state* de l'aposta a *Close*.

- Comprovació apostes en estat *Open*

En aquesta funció es comprova que les apostes en estat *Open* no hagin finalitzat i que la partida d'aquestes apostes tampoc hagi finalitzat.

El primer pas és comprovar que l'aposta hagi finalitzat.

En cas afirmatiu, es realitza una petició a *l'API* de *Riot Games* demanant informació de la partida la qual s'ha realitzat l'aposta, es realitza una transacció des del compte de l'*owner* realitzant el mètode *betClose* de l'*Smart Contract* (tanca l'aposta i es donen els *tokens E-bet* apostats al guanyador) i s'actualitza la base de dades canviant el camp *state* de l'aposta a *Close* i s'introdueix el guanyador al camp *winner*.

En cas negatiu, també es realitza una petició a *l'API* de *Riot Games* demanant informació de la partida la qual s'ha realitzat l'aposta. En aquest cas es comprova que la partida hagi acabat abans que hagi finalitzat l'aposta. Si es dóna aquest cas, es realitza una transacció des del compte de *l'owner* realitzant el mètode *betClose* de l'*Smart Contract* (tanca l'aposta i es donen els *tokens* E-bet apostats al guanyador) i s'actualitza la base de dades canviant el camp *state* de l'aposta a *Close* i s'introdueix el guanyador al camp *winner*.

- Comprovació apostes en estat *Close*

En aquesta funció es mira si les partides de les apostes en estat *Close* ja han acabat i si tenen guanyador. Per realitzar aquesta comprovació es miren totes les apostes que no tinguin cap valor en el camp *winner*. A continuació, es realitza una petició a l'*API* de *Riot Games* demanant informació de la partida la qual s'ha realitzat l'aposta. Si en aquesta informació ja s'informa del guanyador de la partida, es realitza una transacció des del compte de l'*owner* realitzant el mètode *betClose* de l'*Smart Contract* (tanca l'aposta i es donen els *tokens E-bet* apostats al guanyador) i s'actualitza la base de dades canviant el camp *state* de l'aposta a *Close* i s'introdueix el guanyador al camp *winner*.

6.5. *Socket*

En aquest projecte s'ha utilitzat els *Sockets* per poder desenvolupar la funció de xat entre usuaris. Per realitzar-ho, s'ha utilitzat el paquet *Socket.IO*.

Fig. 6.9 Icona de Socket.io

Socket.IO és una llibreria que permet realitzar connexions entre un servidor i un navegador en temps real i bidireccional. A més, perquè el servidor pugui estar pendent de tota la informació que envia el client, *Socket.IO* utilitza *events* que es poden crear tant en el *Front-End* com en el *Back-End*.

En l'*Annex G: Funcions Socket - Xat* es troben totes les funcions que gestionen els xats amb la utilització de la llibreria Socket.IO.

6.6. Seguretat

En aquest projecte s'ha implementat mecanismes de seguretat per xifrar la contrasenya, *Json Web Tokens* per mantenir la sessió d'un usuari, i s'ha hagut de firmar totes les transaccions que s'han llançat a la *Blockchain*.

6.6.1. Xifratge de constrasenyes

Per poder realitzar el xifratge de contrasenyes s'ha decidit treballar amb les funcions resums o *Hash*. Podem aplicar un *Hash* a una contrasenya i posteriorment realitzar una comprovació entre el *hash* i el *hash* emmagatzemat per determinat si l'entrada a la sessió d'usuari és correcte o no.

En aquest projecte, s'ha utilitzat la llibreria *bcrypt*. Aquesta llibreria ens ajuda a crear *Hash* de contrasenyes i, a realitzar comprovacions entre diferents *strings* per poder determinar si una contrasenya és correcte. Al registrar un usuari, es realitza el *hash* de la seva contrasenya (***Fig. 6.10)*** i es guarda a la base de dades. D'aquesta manera evitem guardar contrasenyes a la base de dades en clar.

```
bcrypt.hash(user.password);
```

Fig. 6.10 Creació del hash de la contrasenya

En el moment en què un usuari entra en la seva sessió i introdueix la contrasenya, s'utilitza el mètode *compare* que realitza el *hash* de la contrasenya introduïda i el compara amb el *hash* guardat en la base de dades (***Fig. 6.11)***. En cas que els dos valors siguin idèntics, es procedirà a l'entrada en la sessió d'usuari, i en cas contrari es retorna un error.

```
bcrypt.compare(user.password, userFound.password);
```

Fig. 6.11 Comparació entre el hash creat i el hash guardat

6.6.2. *JSON Web Tokens*

Els J*SON Web Tokens* són un conjunt de mesures de seguretat, que han d'adoptar tant el *Front-End* com el *Back-End*, per tal de realitzar peticions *http* firmades digitalment. El *token* és firmat per una clau que només coneix el servidor. Així, és capaç de verificar l'autenticitat i verificació del *JSON Web Token*.

L'objectiu principal d'aquests *tokens* és propagar la identificació dels usuaris i comprovar la seva identitat al realitzar qualsevol petició *http*.

El J*SON Web Tokens* funcionen de la següent manera:

En el moment en què l'usuari introdueix les seves credencials de manera correcta, el servidor crea el següent *JSON Web Token*:

```
exports.createToken = function (user) {
    let payload = {
        sub: user._id,
        name: user.username,
        iat: moment().unix(),
        exp: moment().add(30, 'days'.unix)
    };

    return jwt.encode(payload, secret);
};
```

Fig. 6.12 Funció de creació del JSON Web Token

Aquest *token* està format per l'identificador i el nom d'usuari de l'usuari i el moment en què es crea el *token* i la data d'expiració del *token*.

A continuació, s'executa el mètode *encode*() de la llibreria *jwt-simple*. Aquest mètode ens permet codificar l'objecte *JSON* creat anteriorment utilitzant un secret.

Posteriorment, aquest *token* és enviat al *Front-End,* que el guarda en el *local storage*. A partir d'aquest punt, l'usuari ha d'introduir aquest *token* en la capçalera de totes les peticions *http* que realitza. Normalment, s'introdueix dins de la capçalera anomenada *Authorization*.

```
let token = req.headers.authorization.replace( searchValue: /['"]+/g, replaceValue: '');

try {
    payload = jwt.decode(token, secret);
    if (payload.exp >= moment().unix){
        return res.status(401).send({
            message: 'Token expired'
        })
    }
} catch(ex){
    return res.status(401).send({
        message: 'Token not valid'
    })
}
```

Fig. 6.13 Funció per comprovar la validesa del JSON Web Token

El servidor, en el moment que rep la petició *http*, comprova la veracitat del token (***Fig. 6.13)***. Aquesta comprovació la realitza executant el mètode *decode*() on introdueix el token rebut des del client i el secret. Si el *token* no és vàlid o ha expirat, s'envia un error i no es realitza la funció per la qual s'havia llençat la petició *http*.

6.6.3. Firma de transaccions

Les transaccions és la manera en què podem comunicar-nos amb una xarxa *Ethereum* modificant l'estat de la xarxa.

Per poder realitzar aquesta modificació d'estats, la transacció enviada ha d'estar signada amb la clau privada del compte *Ethereum* que emet la transacció.

A *Ethereum*, cada compte té una clau pública i una clau privada. A més, si realitzem el hash de la clau pública, obtenim l'adreça *Ethereum* d'un compte. Amb la clau pública, qualsevol usuari pot verificar la identitat i la integritat del missatge. I, amb la clau privada, l'usuari pot signar missatges per poder enviar correctament una transacció. En aquest projecte, s'ha hagut de signar totes les transaccions emeses tant des del *Front-End* com des del *Back-End*.

Com s'ha comentat amb anterioritat, per poder realitzar la firma d'una transacció necessitem la clau privada del nostre compte *Ethereum*. Per gestionar aquests comptes i les seves respectives claus privades, s'ha desenvolupat una *wallet*. Aquesta *wallet* ens permet obtenir de manera senzilla la nostra clau privada a partir de la seguretat que ens aporta la contrasenya de la *wallet*.

El primer pas per signar i enviar una transacció és carregar l'*Smart Contract* al qual ens referim (***Fig. 6.14)***.

```
this.myContract = new web3.eth.Contract(this.abi, this.contractAddress);
```

Fig. 6.14 Carrega del Smart Contract

Per carregar-lo, necessitem conèixer tant *l'ABI* com l'adreça del *Smart Contract*. Posteriorment, s'ha de cridar la funció *encodeABI*() referenciant el mètode del *Smart Contract* que es vol utilitzar (***Fig. 6.15)***.

```
let betCreate = this.myContract.methods.buyTokensPassEthers();
let encodedABI = betCreate.encodeABI();
```

Fig. 6.15 Crida del mètode buyTokensPassEthers() i codificació

Executant aquesta funció codifiquem *l'ABI* per al mètode cridat i obtenim la codificació que necessitem per llançar-lo en una transacció. A més, per realitzar una transacció necessitem conèixer el *nonce* de l'adreça la qual realitza la transacció.

```
let nonce = await this.web3.eth.getTransactionCount(this.selectedAccount.address);
```

Fig. 6.16 Execució del mètode getTransactionCount() per aconseguir el nonce

El *nonce* és el nombre total de transaccions que el compte ha realitzat. Aquest nombre s'incrementa cada vegada que el compte executa una nova transacció, permetent així que la xarxa conegui l'ordre d'execució de transaccions. Per obtenir aquest valor, s'ha d'utilitzar la funció *getTransactionCount* especificant l'adreça amb la qual realitzem la transacció (***Fig. 6.16)***.

El següent pas és crear l'objecte de la transacció.

```
let tx = {
   gas: 1500000,
   gasPrice: '30000000000',
   from: this.selectedAccount.address,
   data: encodedABI,
   chainId: 3,
   to: this.contractAddress,
   nonce: nonce,
   value: this.web3.utils.toWei(''+Math.floor(this.numberTokensBuy / 900, 'ether')
};
```

Fig. 6.17 Creació de l'objecte de la transacció

En aquest objecte, hem d'introduir el gas, el *gasPrice*, l'adreça que executa la transacció, l'adreça a la qual es dirigeix la transacció (si s'executa un mètode d'un *Smart Contract* s'ha d'introduir l'adreça del contracte), el mètode que s'utilitza, l'identificador de la *Blockchain*, el *nonce*, i el valor de *Wei's*.

Per acabar, s'ha d'executar la funció *signTransaction* per signar la transacció, i la funció *sendSignedTransaction* per enviar la transacció signada (***Fig. 6.18)***.

```
this.web3.eth.accounts.signTransaction(tx, this.selectedAccount.privateKey)
.then(signed => {
   this.web3.eth.sendSignedTransaction(signed.rawTransaction);
});
```

Fig. 6.18 Signatura i enviament de la transacció

La funció *signTransaction*() ens permet signar la transacció creada amb la clau privada que tenim emmagatzemada en la *wallet*. Una vegada la transacció està signada, s'executa la funció *sendSignedTransaction*() adjuntant la transacció signada.

Des del *Back-End*, només s'executen les transaccions que realitza *l'owner* del *Smart Contract*. Per llançar la transacció s'ha d'executar tots els mateixos passos anteriorment descrits, amb l'única diferència de la manera en què s'obté la clau privada. Com el compte de *l'owner* el tenim adjuntat en el nostre client de *Blockchain* (*Geth*), hem d'executar les següents comandes per obtenir la clau privada.

```
let keyObject = keythereum.importFromFile(from, datadir);
let privateKey = keythereum.recover(pass, keyObject);
```

Fig. 6.19 Funcions per recuperar la clau privada d'un compte

S'ha d'executar la funció *importFromFile*() on, a partir de l'adreça del compte de l'usuari i el directori on es troba l'arxiu amb les claus del compte, s'obté un objecte que té incrustat la clau privada que necessitem. Però, per obtenir-la, s'ha d'utilitzar la funció *recover*() especificant la contrasenya per desbloquejar el compte. Executant tots aquests passos en detall podem realitzar una transacció amb el format correcte i signada per l'emissor.

CAPÍTOL 7. *FRONT-END*

En aquest capítol s'explica tota la informació que envolta al desenvolupament i resultat del *Front-End* del projecte.

S'explica la navegació utilitzada i les diferents pàgines desenvolupades que conté l'aplicació.

7.1. Navegació:

Per poder utilitzar l'aplicació, és necessari efectuar un registre. En entrar dins la sessió de cada usuari, es pot observar els dos aspectes més influents de navegació que disposa l'aplicació: les *tabs* inferiors i el menú hamburguesa.

7.1.1. *Tabs*

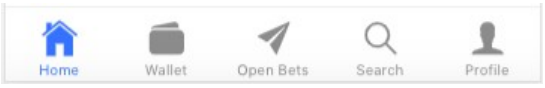


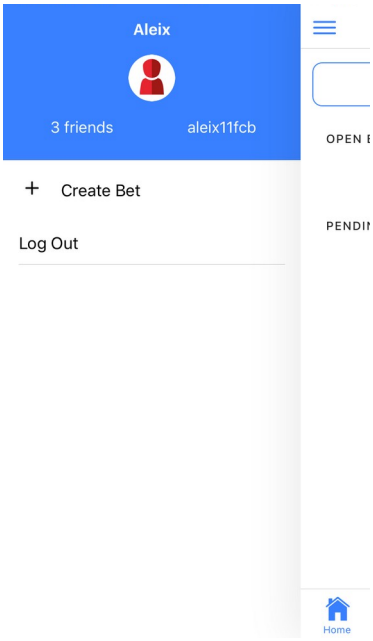
Fig. 7.1 *Tabs* inferiors de l'*app E-bet*

Unes *tabs* són uns botons posicionats en una barra fixa normalment situats a la part inferior de la pantalla. Aquest tipus de navegació s'utilitza en aplicacions que requereixen poques opcions principals, ja que l'espai limita a no tenir més de 5 *tabs* fixes.

Les *tabs* utilitzades en aquesta aplicació es troben en l'apartat inferior de la pantalla en entrar a la sessió d'usuari. En elles podem seleccionar a quina de les 5 pantalles principals podem accedir només amb un simple clic.

- *Tab* 1 - *Home*: Es mostra la informació de les apostes en estat *Open* i *Pending* així com un botó per crear una altra aposta.
- *Tab* 2 - *Wallet*: Es mostra tota la informació relativa a les adreces i els tokens.
- *Tab* 3 - *Open Bets*: Es tracta d'un buscador on es mostren totes les apostes en estat *Pending*.
- *Tab* 4 - *Search*: Es tracta d'un buscador on poder trobar usuaris o Summoners.
- *Tab* 5 - *Profile*: Es mostra tota la informació de l'usuari així com estadístiques d'apostes.

7.1.2. Menú hamburgesa



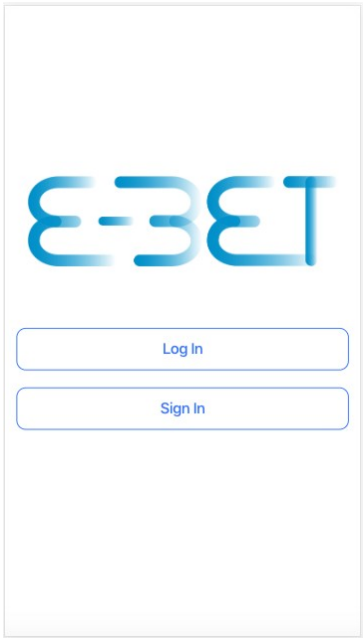
El menú hamburguesa, que s'observa en la ***Fig. 7.2,*** és un menú ocult que s'expedeix clicant a la icona de les 3 barres horitzontals. Aquest tipus de menú és el més utilitzat, ja que permet, amb un simple clic, obrir un apartat en la pantalla on introduir molta informació sense destorbar la informació que es mostra en la pantalla principal.

En menú utilitzat en aquesta aplicació, ens permet accedir al nostre perfil, crear una aposta, visualitzar les notificacions i sortir de la sessió d'usuari.

Fig. 7.2 Menú hamburguesa de l'*app E-bet*

7.2. Pàgines

7.2.1. Pàgina d'inici



Aquesta pàgina (***Fig. 7.3)*** és la primera a mostrar-se en obrir l'aplicació. És una simple pàgina on hi ha el logo de l'aplicació i dos botons per accedir a la pàgina de registre o la pàgina d'entrada.

Fig. 7.3 Pàgina d'inici de l'app E-bet

7.2.2. Pàgina de registre

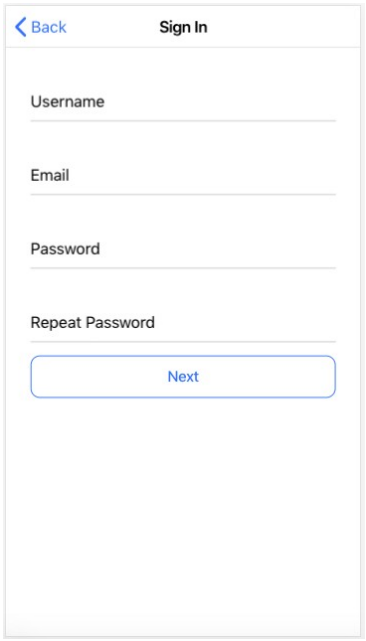


Fig. 7.4 Pàgina de registre de l'app E-bet

Aquesta pàgina (***Fig. 7.4)*** és la primera pàgina que s'ha de completar per poder registrar-se en l'aplicació. En ella es demana el nom d'usuari, *l'email* i la contrasenya a utilitzar per poder accedir dins l'aplicació. Una vegada completat totes les dades, el botó *Next* permetrà accedir a la pàgina de registre de la *wallet*.

7.2.3. Pàgina de registre de la Wallet

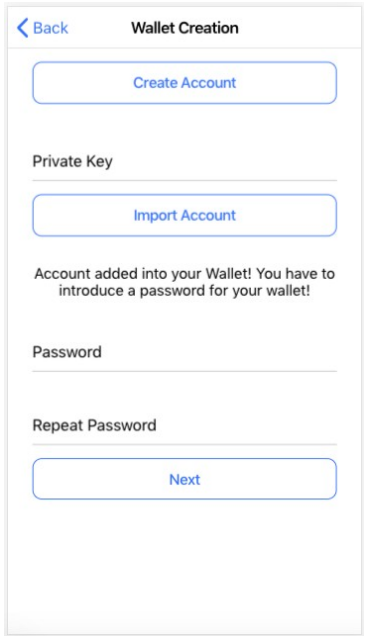
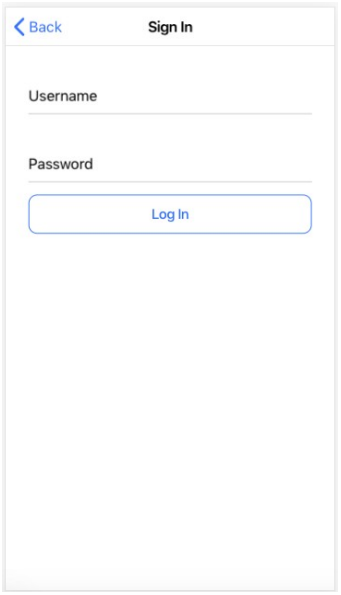


Fig. 7.5 Pàgina de registre de la *Wallet* de *l'app E-bet*

Aquesta pàgina (***Fig. 7.5)*** és la segona i última pàgina que s'ha de completar per poder registrar-se en l'aplicació. En entrar es crearà una *wallet* en el dispositiu utilitzat i es donarà de crear un compte *Blockchain* o d'importar-la utilitzant la clau privada. Una vegada registrar un compte, es demanarà introduir una contrasenya que s'utilitzarà per desbloquejar la *wallet* i per realitzar totes les futures transaccions. Una vegada completat totes les dades, el botó *Next* permetrà accedir a la pàgina *Tab* 1 dins la sessió d'usuari.

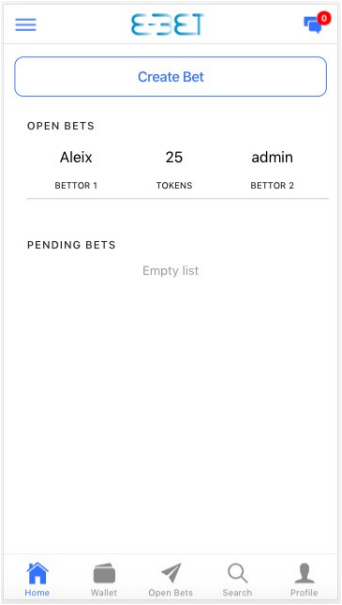
7.2.4. Pàgina d'Entrada



Aquesta pàgina (***Fig. 7.6)*** es demana el nom d'usuari i la contrasenya per poder accedir dins l'aplicació. Una vegada completat totes les dades, el botó Log In permetrà accedir a la pàgina Tab 1 dins la sessió d'usuari.

Fig. 7.6 Pàgina d'entrada de l'*app E-bet*

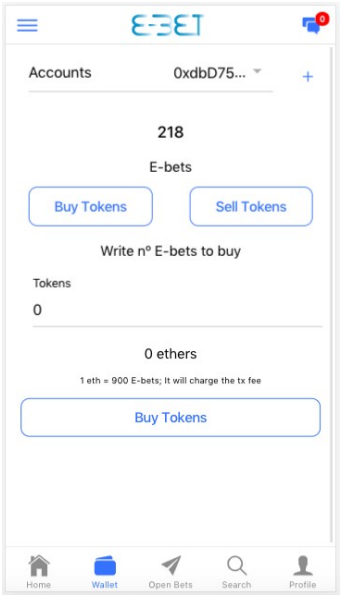
7.2.5. Pàgina Tab 1



Aquesta pàgina (***Fig. 7.7***) és la *tab Home*, la primera pàgina que es mostra en accedir dins la sessió d'usuari. En ella, hi ha el botó principal per crear un aposta accedint a la pàgina de creació d'aposta. A continuació, es mostra una llista amb les apostes obertes i una altra llista amb les apostes pendents que l'usuari disposa.

Fig. 7.7 Pàgina *Tab* 1 de l'*app E-bet*

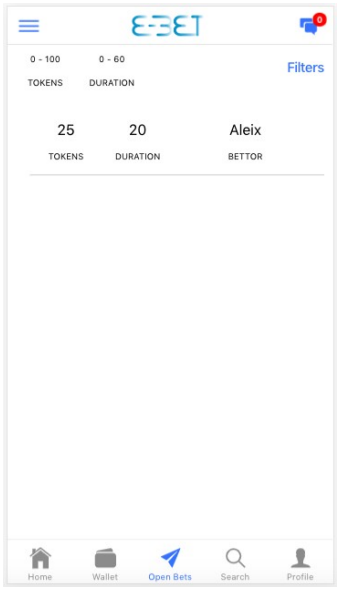
7.2.6. Pàgina Tab 2



Aquesta pàgina (***Fig. 7.8***) és la *tab Wallet*. Si no s'ha accedit amb anterioritat, el primer pas que demana és introduir la contrasenya de la *wallet* per desbloquejar-la. En aquesta pàgina s'informa de les adreces que disposa l'usuari amb la quantitat de *tokens* que té cadascuna. A més, dóna l'opció de crear o importar una altra adreça al compte de l'usuari. També, és la pàgina on es poden comprar i vendre els *tokens* que es desitgin intercanviant-los per *Ethers*.

Fig. 7.8 Pàgina *Tab* 2 de l'*app E-bet*

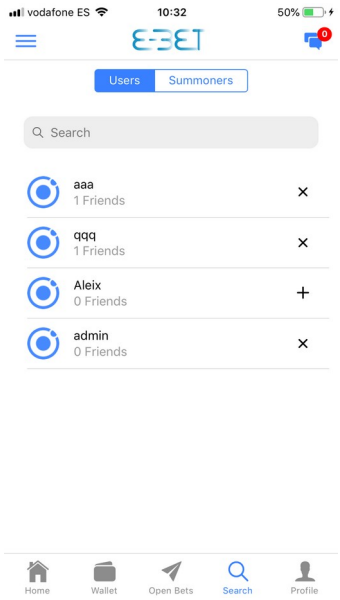
7.2.7. Pàgina Tab 3



Aquesta pàgina (***Fig. 7.9***) és la *tab Open Bets*. En ella, es mostren les apostes en estat *Pending* que es poden acceptar. A més, en la part superior hi ha un filtre que ens permet restringir la cerca segons un mínim o un màxim de *tokens*, un mínim o una màxima duració de l'aposta, el nom del *Summoner* al qual s'ha creat la partida, o el nom d'usuari del creador de l'aposta.

Fig. 7.9 Pàgina *Tab 3* de l'*app E-bet*

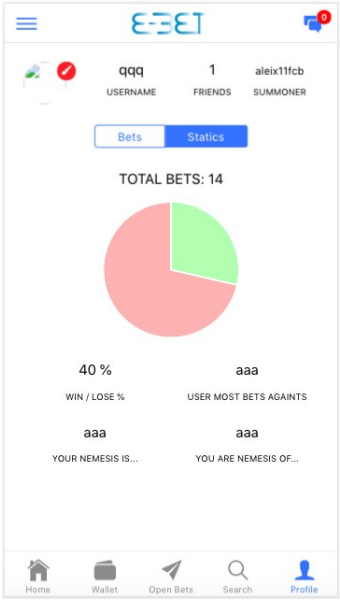
7.2.8. Pàgina Tab 4



Aquesta pàgina (***Fig. 7.10***) és la *tab Search*. En ella, es poden buscar usuaris de l'aplicació o *Summoners* del *League of Legends*. Al buscar usuaris ens permet afegir-los o borrar-los com a amics, i al buscar *Summoners* ens permet afegir-los o treure'ls com a *Summoner* preferit.

Fig. 7.10 Pàgina *Tab 4* de l'*app E-bet*

7.2.9. Pàgina Tab 5



Aquesta pàgina (***Fig. 7.11***) és la *tab Profile*. En ella, es mostra tota la informació pròpia de l'usuari. Ens permet saber el nombre d'amics que tenim, el *Summoner* preferit, les apostes realitzades i les estadístiques d'apostes guanyades i perdudes, el total d'apostes realitzades, l'usuari contra qui més vegades s'ha apostat, qui és el teu usuari *nemesis* i de l'usuari a qui tu ets el seu *nemesis*.

Fig. 7.11 Pàgina *Tab 5* de l'*app E-bet*

7.2.10. Pàgina de creació de l'aposta

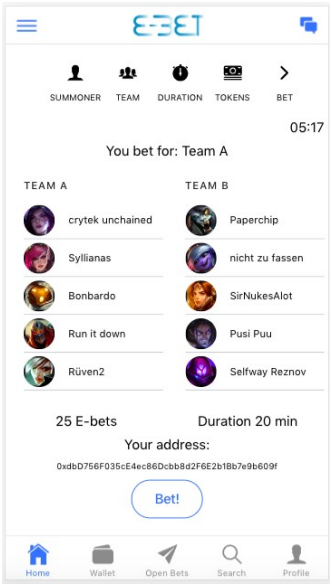


Fig. 7.12 Pàgina de creació de l'aposta de l'*app E-bet*

Aquesta pàgina (***Fig. 7.12***) es troben tots els passos per realitzar una aposta. El primer pas és introduir el nom d'un *Summoner* que estigui jugant una partida. A continuació, es mostra tota la informació de la partida que el *Summoner* està jugant, tant el minut exacte de joc com la informació dels *Summoners* que conformen cada equip i els personatges que utilitza cada *Summoner*. En aquest apartat, l'usuari ha de seleccionar per quin equip vol apostar. Seguidament, ha de marcar la duració que vol que tingui l'aposta així com la quantitat de *tokens E-bet* que vol apostar. Finalment, i abans de crear l'aposta, es mostra un resum amb tota la informació de la partida i de l'aposta que es vol realitzar.

7.2.11. Pàgina aposta

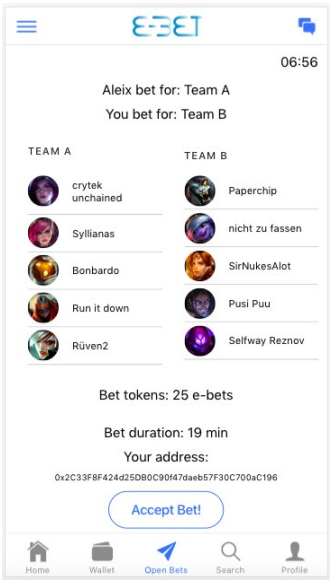


Fig. 7.13 Pàgina aposta de l'*app E-bet*

Aquesta pàgina (***Fig. 7.13***) ens mostra tota la informació disponible d'una aposta. A la part superior de la imatge es mostra el temps de duració de la partida juntament amb l'equip pel qual ha apostat cada usuari. Seguidament, es mostra els *Summoners* que conformen cada equip i els personatges que utilitza cada *Summoner*. A continuació, es mostren els *tokens* apostats, la duració de l'aposta i l'adreça que l'usuari ha utilitzat per realitzar l'aposta.

7.2.12. Pàgina llistat de xats

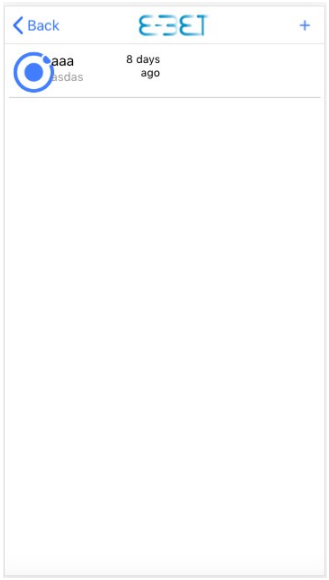


Fig. 7.14 Pàgina llistat de xats de l'*app E-bet*

En aquesta pàgina (***Fig. 7.14***) es mostra una llista amb els xats oberts amb els altres usuaris. Al clicar sobre un usuari, s'obre la pàgina Xat on es mostra la conversa amb l'usuari seleccionat. A més, en el menú superior hi ha un botó que obra un modal que permet buscar l'usuari al qual es vol obrir una conversa.

7.2.13. Pàgina xat

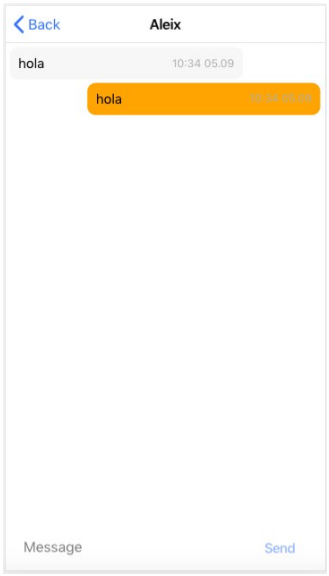


Fig. 7.15 Pàgina xat de l'*app E-bet*

En aquesta pàgina (***Fig. 7.15***) es mostra la conversa amb l'usuari seleccionat. En ella, es mostren tots els missatges enviats entre aquests dos usuaris, així com el moment exacte en què s'ha enviat el missatge. A més, en la part inferior es permet introduir un nou missatge que es vol enviar a l'altre usuari.

7.3. Seguretat

A més dels sistemes de seguretat comentats en l'apartat de *Back-End* i que també s'utilitzen en el *Front-End* com els *tokens* i la signatura de transaccions, en el *Front-End* trobem la *Wallet*.

7.3.1. *Wallet*

Una *Wallet* a *Ethereum* és un sistema que gestiona els comptes i balanços d'un usuari, i que permet interaccionar amb qualsevol plataforma o aplicació que estigui construïda per sobre de la xarxa *Ethereum*.

En aquest projecte, s'ha desenvolupat una *wallet* utilitzant el mètode *wallet* que proporciona la llibreria *web3* i una sèrie de funcions desenvolupades en l'*Smart Contract Bet.sol* i *ERC20.sol*.

En el moment en què un usuari efectua un registre, ha de registrar una *wallet* creant un compte o important-lo amb la clau privada i introduint una nova contrasenya. Aquesta contrasenya s'ha d'introduir per desbloquejar la *wallet* i per realitzar qualsevol transacció amb la *Blockchain*. La *wallet* creada es guarda en memoria, així es pot accedir a ella de forma ràpida i segura.

A més, s'ha creat una pàgina on poder gestionar tots els comptes i la quantitat de *tokens E-bet* que té cada compte. En aquesta pàgina, a més, es poden crear i importar comptes, i comprar i vendre *tokens E-bet* per *Ethers*.

Finalment, i com a una de les funcions principals, aquesta *wallet* ens permet gestionar i emmagatzemar la clau privada dels comptes. Aquesta clau privada és el paràmetre que necessitem a l'hora de signar una transacció que volem realitzar.

CAPÍTOL 8. CONCLUSIONS I LÍNIES FUTURES

En aquest apartat, s'explicaran tots els passos tant obligatoris com opcionals que s'han de realitzar en un futur per tal d'enllestir l'aplicació. A més, s'exposen les conclusions obtingudes al realitzar el projecte.

8.1. Línies futures

En el punt actual on es troba l'aplicació, els usuaris poden realitzar totes les funcions principals que es troben descrites al llarg d'aquest document. Però, per tal que l'aplicació pugui tirar endavant i es pugui treure al mercat s'han de realitzar una sèrie de passos.

- **Contactar amb *Riot Games* per poder publicar l'aplicació.** En unes de les últimes normes descrites per *Riot Games*, es prohibia l'ús de la seva API per a qualsevol projecte on s'utilitzes alguna xarxa *Blockchain*. Per aquest motiu, és necessari contactar directament amb ells i arribar a un acord per tal de publicar l'aplicació desenvolupada.

- **Introducció de publicitat**. Per tal d'incrementar els ingressos, tal com s'ha comentat en l'apartat de negocis de l'aplicació, s'ha d'introduir publicitat en algunes zones de l'aplicació. Per realitzar-ho, es pot utilitzar el *plugin AdMob Pro* que ens permet obtenir publicitat amb unes poques línies de codi en *Ionic*.

- **Regal de *tokens E-bet* al registrar-se i al convidar qualsevol altre usuari**. Per tal d'incentivar la participació i el registre d'usuaris, s'ha estudiat regalar una quantitat petita de *tokens E-bet* en el moment de registrar-se a l'aplicació. També, s'efectuarà aquest petit regal en el moment que un usuari comparteixi i provoqui que una altra persona es registri. En aquest cas, els dos usuaris rebran una petita recompensa en forma de *tokens E-bet*.

- **Seguretat**. S'han d'incloure alguns mecanismes més de seguretat. S'hauria d'implementar *https* en totes les peticions que es realitzin. Per realitzar-ho, s'ha de crear un certificat digital pel nostre servidor i implementar la llibreria *https* que ens permet crear un servidor *https* a partir de l'arxiu *.crt* i l'arxiu *.key*.

- **Disseny**. Per tal de millorar l'aspecte de l'aplicació i la seva jugabilitat, s'ha de realitzar un disseny nou i actual. A més, si es crea una marca senzilla però reconeixible, l'aplicació s'expandiria de manera més ràpida.

- **Executar les transaccions i esperar la confirmació en segon pla.** Per tal de millorar la jugabilitat de l'aplicació, a l'hora de llençar una transacció es pot esperar la seva confirmació en segon pla, així els usuaris podrà seguir utilitzant l'aplicació.

- **Notificacions per expandir l'aposta**. Per tal d'expandir la creació de l'aposta, s'enviaria una notificació a tots els amics d'un usuari en el moment que aquest crea una aposta.

- **Compartir una aposta**. Per tal de millorar la jugabilitat i permetre que una aposta s'accepti amb més facilitat, es podrien desenvolupar mecanismes per compartir l'aplicació. Es podria compartir mitjançant xarxes socials, creant un codi QR o mitjançant notificacions.

- **Apostes privades**. Per tal de millorar la jugabilitat i la privacitat de l'aplicació, es podrien crear unes apostes privades on s'hauria d'escriure una contrasenya per poder acceptar l'aplicació. D'aquesta manera es restringeix que un usuari qualsevol accepti una aposta que l'usuari creador hauria aparaulat amb un tercer usuari.

- **Apostes visibles i no visibles en el perfil**. Per tal de millorar la privacitat i la intimitat dels usuaris, al crear l'aposta es podria configurar la seva privacitat. Així, si un usuari no vol que es mostri una aposta en el seu perfil, aquesta quedarà amagada i només accessible pels apostants.

8.2. Conclusions

En aquest projecte s'ha realitzat el desenvolupament de l'aplicació *E-bet*. Aquesta aplicació permet executar apostes un contra un del joc *League of Legends*. Per tal de desenvolupar el projecte, s'ha hagut de programar tant els Smart Contracts, el *Back-end* i el *Front-End*.

El codi font desenvolupat es pot consular en els següents repositoris de *Github*:

- *Back-End*: https://github.com/Aleix11/server-tfg
- *Front-End*: https://github.com/Aleix11/client-tfg-ionic

Amb el desenvolupament del *Smart Contract Bet.sol*, s'ha permès realitzar una gestió dels estats de les apostes senzilla i visual, que permet conèixer en tot moment el seu estat i els participants. A més, a l'estar cada transacció i cada canvi d'estat introduït en la *Blockchain*, podem assegurar-nos en tot moment quan una transacció es enviada cap al contracte i des de quina adreça s'ha cridat. També, amb l'ús de la funció *require*() s'ha limitat molt el llançament de mètodes del *Smart Contract*. Aquest fet significa que, si no es compleixen una serie de requisits a l'hora de llançar una transacció, aquesta es descarta. Per una altra part, cada transacció de canvi d'estat d'una aposta disposa d'una crida a una funció del *Smart Contract ERC20.sol*. Aquest *Smart Contract* ha facilitat de gran manera la creació i transferència de *tokens* entre adreces.

Dins també del *Smart Contract Bet.sol* es troben les funcions de compra-venda de *tokens*. Aquestes funcions permeten l'intercanvi de manera segura entre *tokens E-bet* i *Ethers*. A més, amb la creació dels *mappings* que ens permeten conèixer en tot moment si l'usuari té algun intercanvi pendent de rebre per part

de *l'owner*. Al necessitar que es compleixin les condicions establertes en el contracte, i el fet que per realitzar la compra-venda dels *tokens E-bet* es necessiti executar una transferència des d'un compte d'un usuari i des del compte *owner*, ens permet executar de manera segura aquest tipus d'intercanvis.

Pel que fa al desenvolupament del *Back-End*, la connexió amb *l'API* de *Riot Games* permet tenir en qualsevol moment informació real de les partides que es juguen. A més, amb la utilització del *Cron* podem realitzar les comprovacions dels estats de les apostes de manera continuada en un període de temps molt curt. D'aquesta manera obtenim informació de les partides de manera ràpida i modificar la informació de les apostes gairebé en temps real.
Com el nombre de peticions a *l'API* de *Riot Games* està limitat, per tal d'evitar l'enviament repetit de peticions es guarda a la base de dades tota la informació d'una partida que es necessita a l'hora de realitzar una aposta. D'aquesta manera, en el moment en què es vol buscar una aposta, es realitza una recerca ràpida a la nostre base de dades, evitant així, l'enviament massiu de peticions i provocant una reducció del temps de resposta cap al *Front-End*.

En el *Front-End*, s'ha creat una aplicació hibrida que permet executar l'aplicació en qualsevol sistema operatiu (*Android*, *IOs* i *Windows Phone*). D'aquesta manera l'aplicació es accessible per a qualsevol persona que disposi d'un *Smartphone*.

Dins el dispositiu mòbil es crea una *Wallet* en el moment en què un usuari es registra o entra en la seva sessió. D'aquesta forma es gestionen de manera segura totes les transaccions executades directament des del client signant-les amb la clau privada del compte de l'usuari. Aquest fet ens permet complir un objectiu de les xarxes *Blockchain*, la distribució i la no centralització a l'hora de realitzar i executar una transacció per part d'un usuari.

Pel que fa a l'obtenció de beneficis, l'ús de l'aplicació marcarà els ingressos que s'obtindran. Com més usuaris participin en l'aplicació, més apostes es realitzaran i més compres i vendes de *tokens E-bet* s'executaran, més comissions es produiran, més visualitzacions d'anuncis es faran i més beneficis s'obtindran.

En conclusió, l'aplicació *E-bet* proposa un nou enfocament a l'hora de realitzar apostes. Amb la introducció de la possibilitat d'apostar en partides amateurs, l'expansió de l'aplicació es pot realitzar de manera ràpida, ja que s'executen apostes tant de famosos com de coneguts personals. Realitzant unes millores i arribant a una sèrie d'acords, l'aplicació es podria llençar al mercat i obtenir beneficis gràcies als aspectes anteriorment descrits.

BIBLIOGRAFIA

[1] Cánovas Esteban, A (Juny, 2018). *Xarxes Blockchain – Ethereum*

[2] L. Muñoz Tapia, J - Hernández Serrano, J (Gener 2019). *Blockchain*

[3] L. Muñoz Tapia, J - Hernández Serrano, J (Gener 2019). *Solidity (Part I)*

[4] L. Muñoz Tapia, J - Hernández Serrano, J (Gener 2019). *Solidity (Part II)*

[5] L. Muñoz Tapia, J - Hernández Serrano, J (Gener 2019). *Contracts*

[6] *Modern Cryptography – Network Security EETAC, UPC.*

[7] Dr. Wood, G (Agost, 2019). *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER [PDF]*. Disponible a: https://ethereum.github.io/yellowpaper/paper.pdf

[8] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System [PDF]. Disponible a: https://bitcoin.org/bitcoin.pdf

[9] *Ethereum*. Disponible a: https://www.ethereum.org

[10] *Ethereum – Remix, Solidity IDE*. Disponible a: https://remix.ethereum.org

[11] *Web3.js – Etheureum JavaScript Api*. Disponible a:

https://web3js.readthedocs.io/en/v1.2.1/

[12] *ERC-20. ¿Qué signicfica exactamente? ¿Cuáles son los Tokens y monederos ERC-20?* Disponible a:

https://www.miethereum.com/smart-contracts/erc20/

[13] Li, K (Setembre, 2018) *Ethereum's ERC-20 Tokens Explained, Simply.* Disponible a: https://hackernoon.com/ethereums-erc-20-tokens-explained-simply-88f5f8a7ae90

[14] William, M (Maig, 2018) *Explicación de los tokens ERC-20.* Disponible a: es.cointelegraph.com/explained/erc-20-tokens-explained

[15] Agrawal, H (Agost, 2019), *What is an ERC20 Token?* Disponible a: https://coinsutra.com/what-is-erc20-token/

[16] *Yang, A (Juny, 2018) How do I sign transactions with Web3? Disponible a:* https://medium.com/finnovate-io/how-do-i-sign-transactions-with-web3-f90a853904a2

[17] de Miranda Colaço, A (Abril, 2018) Signing and making transaction on Ethereum using web3.js. Disponible a:

https://medium.com/coinmonks/signing-and-making-transactions-on-ethereum-using-web3-js-1b5663207d63

[18] *DAPPS. ¿Qué son y para qué sirven las Aplicaciones Decentralizadas o Đapps? Disponible a:* www.miethereum.com/smart-contracts/dapps/

[19] Criptografía asimétrica. Disponible a:

https://academy.bit2me.com/que-es-criptografia-asimetrica/

[20] Wikipedia (Agost, 2019). *Blockchain*. Wikipedia, The Free Encyclopedia. Disponible a: https://en.wikipedia.org/wiki/Blockchain

[21] Wikipedia (Agost, 2019). *Ethereum*. Wikipedia, The Free Encyclopedia. Disponible a: https://es.wikipedia.org/wiki/Ethereum

[22] Wikipedia (Agost, 2019). *ERC-20*. Wikipedia, The Free Encyclopedia. Disponible a: https://en.wikipedia.org/wiki/ERC-20

[23] Wikipedia (Agost, 2019). *Cryptographic hash function*. Wikipedia, The Free Encyclopedia. Disponible a:

https://en.wikipedia.org/wiki/Cryptographic_hash_function

[24] Wikipedia (Agost, 2019). *Base de datos distribuida*. Wikipedia, The Free Encyclopedia. Disponible a:

https://es.wikipedia.org/wiki/Base_de_datos_distribuida

[25] Wikipedia (Juliol, 2019). *Prueba de participación*. Wikipedia, The Free Encyclopedia. Disponible a:

https://es.wikipedia.org/wiki/Prueba_de_participación

[26] Wikipedia (Agpst, 2019). *Proof of authority*. Wikipedia, The Free Encyclopedia. Disponible a:

https://en.wikipedia.org/wiki/Proof_of_authority

[27] *Ionic Framework docs.* Disponible a: https://ionicframework.com/docs

[28] *Truffle Suite docs.* Disponible a: https://www.trufflesuite.com/docs

[29] Riot Games. *Riot Games Developer.* Disponible a:

https://developer.riotgames.com

ANNEXOS

TÍTOL DEL TFG: E-bet: Joc distribuït i tokenitzat amb *Blockchain*

TITULACIÓ: Grau en Enginyeria Telemàtica

AUTOR: Aleix Cánovas Esteban

DIRECTOR: Juan Hernández Serrano

DATA: 5 de setembre del 2019

ANNEX A - GUIA D'USUARI

En aquest apartat s'explicaran tots els passos a realitzar per tal de crear i acceptar una aposta.

Per poder anar a la pantalla on crear una aposta, primer l'usuari ha d'anar a la *Tab Home.* Clicant el botó de *Create Bet* que es pot observar en la ***Fig. A.1*** s'accedeix a la pàgina de crear aposta. A més, també es podrà accedir obrint el menú hamburguesa i clicant també el botó *Create Bet*.

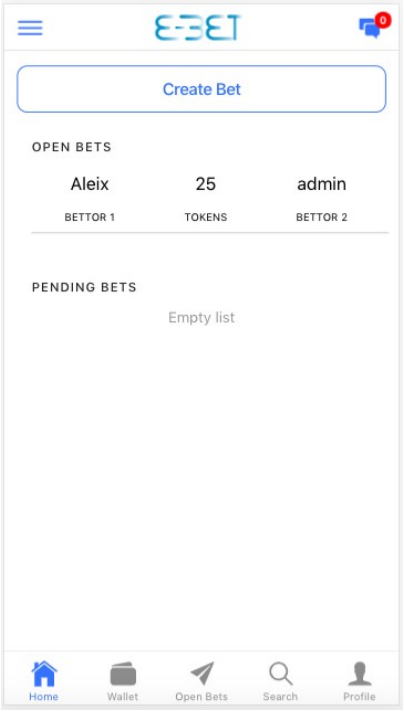


Fig. A.1 *Tab Home* de l'*app E-bet*

Una vegada s'ha accedit a la pàgina de crear aposta, com s'observa en la ***Fig. A.2***, s'ha d'introduir el nom del *Summoner* que està realitzant una partida.

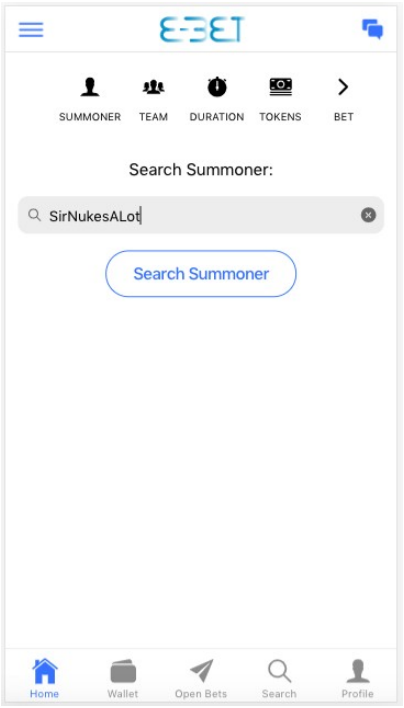


Fig. A.2 Buscador de *summoners* de l'*app E-bet*

A continuació, s'ha d'escollir l'equip al qual es vol apostar. Com s'observa en la ***Fig. A.3***, es mostren la informació dels dos equips, amb els personatges de cada *Summoner* i el temps de partida.

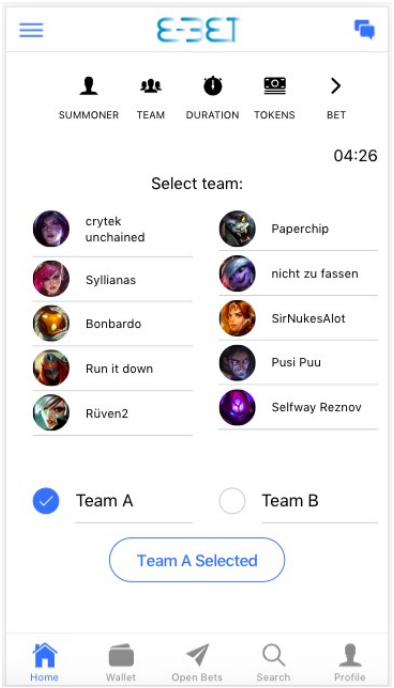


Fig. A.3 Informació dels equips d'una aposta de l'*app E-bet*

Després, i com s'observa en la ***Fig. A.4****,* s'ha d'escollir la duració de l'aposta. Encara que es pot escollir una duració entre 1 minut i 60, si una partida acaba abans que acabi la duració de l'aposta, aquesta també es tanca.

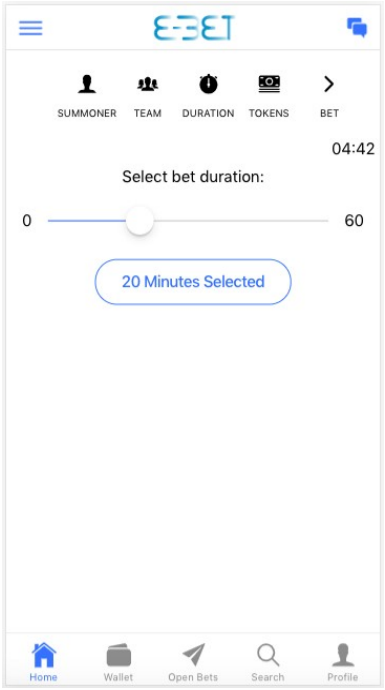


Fig. A.4 Seleccionador de duració d'aposta de l'*app E-bet*

Seguidament, i com s'observa en la Fig. A.5, s'ha d'escollir la quantitat de tokens E-bet a apostar. Es pot escollir entre aposta 1 o tots els tokens E-bet disponibles per l'usuari.

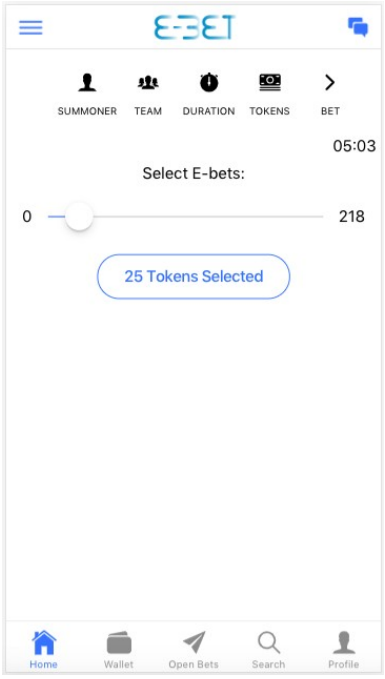


Fig. A.5 Seleccionador de *tokens E-bet* de l'*app E-bet*

Per acabar amb la creació de l'aposta, i tal com s'observa en la ***Fig. A.6***, es mostra un resum amb tota la informació seleccionada. Per tal de completar la creació de l'aposta, s'ha de clicar al botó *Bet!*.

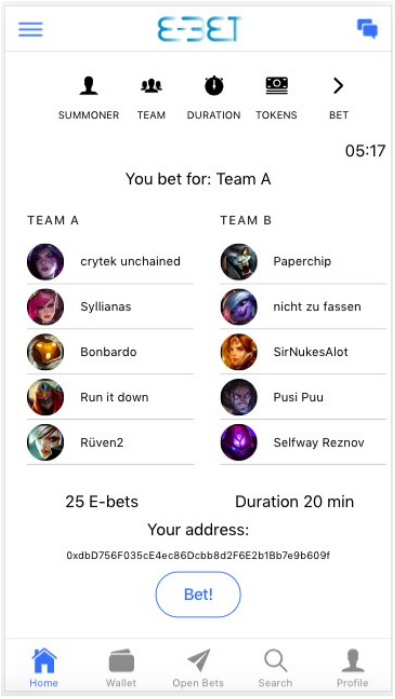


Fig. A.6 Resum de l'aposta de l'*app E-bet*

Finalment, s'ha d'introduir la contrasenya de la *wallet* per tal de poder llançar l'aposta.

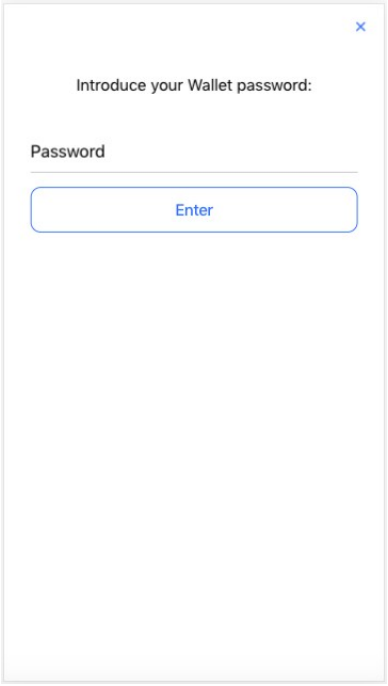


Fig. A.7 Pàgina per introduir constrasenya de la *wallet*

El següent pas per tal que una aposta es crei i s'accepti, és que un altre usuari realitzi l'acceptació de l'aposta. Per fer-ho i tal com s'observa en la ***Fig. A.8,*** l'usuari ha d'anar a la *Tab Open Bets* on podrà observar totes les apostes en estat *Pending*.

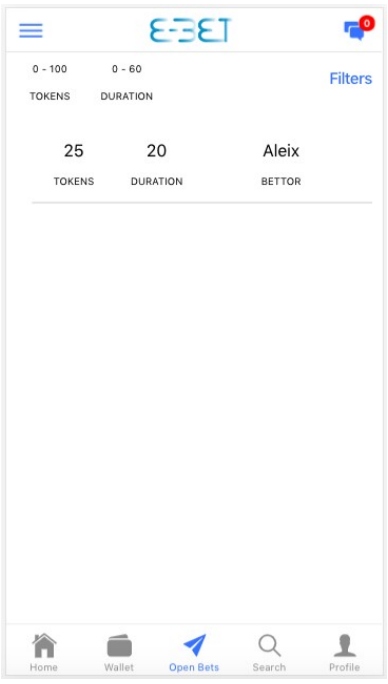


Fig. A.8 *Tab Open Bets*

Al clicar sobre qualsevol aposta, i tal com s'observa en la ***Fig. A.9***, s'obre una pàgina on hi ha tota la informació de l'aposta i el botó *Accept Bet!,* que permet acceptar una aposta.

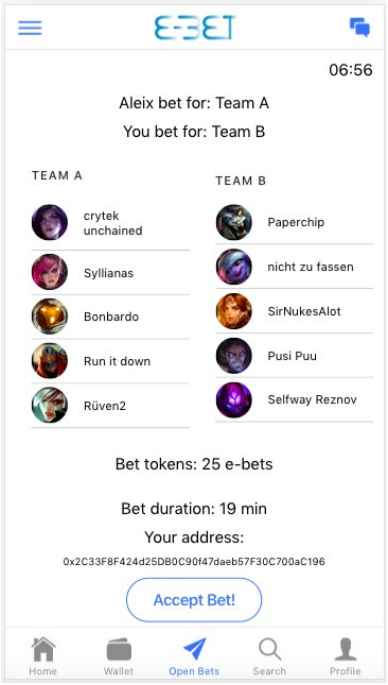


Fig. A.9 Pàgina d'informació de l'aposta

Finalment, i de la mateixa manera que al crear una aposta, s'ha d'introduir la contrasenya de la *wallet* per tal de poder acceptar-la.

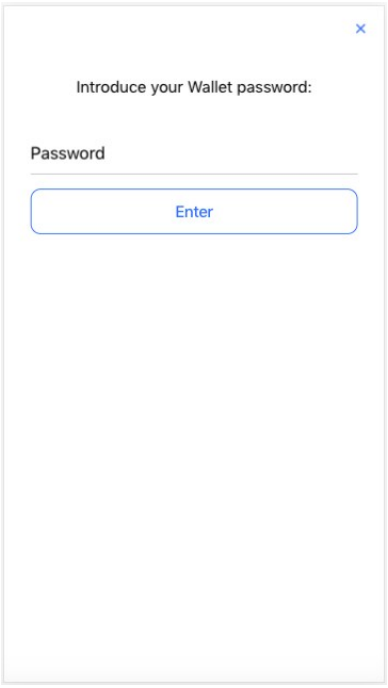


Fig. A.10 Pàgina per introduir constrasenya de la wallet

ANNEX B - MECANISMES DE CONSENS - *BLOCKCHAIN*

A continuació, s'expliquen els mecanismes de consens més famosos i més utilitzats.

- *Proof of Work*: És el mecanisme de consens original i més famós de les xarxes *Blockchain*. En aquest mecanisme, apareix la figura dels miners.

  Els miners són nodes que es dediquen a resoldre problemes matemàtics (habitualment, funcions *hash*). Aquests miners competeixen entre ells per completar les transaccions de la xarxa *Blockchain* i a canvi reben una petita recompensa. Una vegada resolen aquest problema matemàtic, obtenen una solució (anomenada *hash*) i verifiquen les transaccions.

  Els beneficis principals d'aquest sistema són la defensa sobre els atacs de denegació de servei i la nul·la rellevància dels usuaris amb grans sumes de diners envers la gestió de la xarxa. L'atac *DoS* és possible en aquestes xarxes, però l'esforç computacional i temporal són tan alts que crearien moltes més despeses que beneficis per l'atacant. A més, no importa la quantitat de diners que tingui un compte d'usuari, l'important és tenir un gran poder computacional per poder minar i crear nous blocs.

  Aquest mecanisme de consens és l'utilitzat a les *Blockchain* més famoses i generalment públiques, com *Bitcoin*, *Ethereum*, i moltes altres.

- *Proof of Elapsed Time*: Es tracta d'un mecanisme de consens utilitzat normalment en les xarxes privades on es necessita el permís d'algun administrador. Les *Blockchain* que utilitzen aquest mecanisme de consens estan basades en un sistema de loteria justa, on es decideix els drets dels miners o dels guanyadors de blocs de la xarxa.

  El funcionament d'aquest algoritme és el següent: Cada node participant necessita esperar un període de temps escollit de forma aleatòria. El primer node que se li acabi aquest temps és el designat com a guanyador del nou bloc. El guanyador és l'encarregat de proposar un nou bloc, i el procediment es torna a repetir.

  Aquest mecanisme ha d'assegurar que el temps d'espera sigui aleatori i que el guanyador hagi completat tot el temps d'espera.

  Aquest mecanisme de consens està creat per *Intel* i es pot utilitzar a la B*lockchain HyperLedger*.

- *Proof of Authority*: Aquest mecanisme de consens no es basa en la resolució de problemes matemàtics (mineria), ni en un temps aleatori. En aquest tipus de *Blockchain* existeixen uns nodes amb una autoritat que els hi permet crear blocs nous, verificar-los i assegurar la xarxa *Blockchain*.

Per poder verificar una transacció, aquesta ha de ser verificada per la majoria dels nodes amb autoritat.

Aquest tipus de *Blockchain* poden ser més centralitzades que les altres, i poden gestionar un nombre més elevat de transaccions per segon, ja que no depenen del cost computacional per resoldre un problema matemàtic (mineria).

Aquest mecanisme de consens s'utilitza en xarxes de test com *Rinkeby* i *Kovan*.

- *Proof of Stake*: Es tracta d'un mecanisme de consens que estableix que un node pot validar blocs d'acord amb la quantitat de monedes que té.

  Quan un node té moltes unitats d'una criptomoneda està interessat en la supervivència i el bon funcionament d'aquesta. Per tant, són els més indicats en tenir la gran responsabilitat de protegir-la. És per aquest motiu que el mecanisme premia aquests nodes amb una menor dificultat per trobar blocs.

  A diferència d'altres mecanismes de consens, els miners no reben una recompensa en validar un bloc. Com que no existeix aquesta recompensa, els miners guanyen el valor de la taxa de la transacció.

  Aquest mecanisme de consens ha estat creat com a alternativa a *Proof of Work*. El gran avantatge respecte als altres mecanismes de consens és que es pot tenir un grau de seguretat equivalent a *Proof of Work* però amb una despesa energètica menor i sense la necessitat de crear una competició per minar.

  En contra, aquest mecanisme de consens necessita que el miner mantingui la seva *wallet* oberta. A més, es poden provocar atacs del 51%. Quan un node/grup de nodes té el 51% de la participació de la xarxa pot tenir el control de la cadena. Amb aquest control, la xarxa deixa d'estar distribuïda i aquest node pot validar o descartar els blocs que vulgui.

ANNEX C - NORMATIVA RIOT GAMES

1. Validació de les peticions

Abans de realitzar cada petició, hem de validar que tots els caràcters inclosos al buscar un nom d'un *summoner* siguin caràcters Unicode, dígits, espais, barra baixes i punts. Expressió per validar aquests noms de *summoner*: ^[0-9\\p{L} _\\.]+$

2. Els criteris

Els criteris que utilitzen els moderadors per determinar si una aplicació és aprovada per una clau de producció són els següents:

- Que el projecte estigui acabat o gairebé acabat. A més, el projecte ha de tenir una pàgina d'inici on els usuaris poden obtenir més informació.
- Que es pugui veure clar l'ús de *l'API* en el projecte.
- Que el projecte no entri en conflicte en cap de les polítiques de Riot Games.
- Que, si el projecte té un sistema d'autentificació, s'ha de proporcionar una mostra d'inici de sessió pels moderadors de *Riot Games*. A més, si el projecte requereix l'ús d'un *smartphone* o *iPad*, la pàgina web ha de proporcionar documentació suficient per avaluar el projecte. Aquesta documentació ha d'incloure imatges i descripcions de l'aplicació, a més de les entrades dels usuaris i les sortides que mostren les dades recopilades.

Aquests criteris són d'obligat compliment per tal de poder treure l'aplicació en el mercat. En cas de violar algun d'aquests criteris, s'ha de realitzar una consulta directament als moderadors de *Riot Games* per poder demanar l'aprovació de l'aplicació.

3. Claus de *l'API*

Existeixen dos tipus de claus *d'API* segons les privacitats del projecte:

- Desenvolupament i claus *d'API* internes:

Clau *API* per projectes no públics. S'utilitza per desenvolupar un prototip per un projecte futur. El límit de peticions per una *development key* és:

20 peticions cada 1 segon.

100 peticions cada 2 minuts.

- Claus *d'API* de producció

El límit de peticions per una clau en producció és:

3.000 peticions cada 10 segon.

180.000 peticions cada 10 minuts.

4. Polítiques generals

El projecte a desenvolupar, no ha de:

- Violar la llei.
- Utilitzar cap dels logos oficials.
- Referenciar que el projecte a desenvolupar és una associació o està aprovat per *Riot Games*.
- Ser publicat si no s'assegura adequadament *l'API key*.
- Utilitzar una *development key* si, aquest, és públic i/o accessible per a la comunitat.
- Utilitzar una *production key* si aquesta *API key* s'utilitza en altres projectes. (1 projecte, 1 *API key*).
- Comprometre la integritat del joc, o crear un avantatge injust per als jugadors.
- Carregar diners per a l'aplicació o proporcionar accés exclusiu a usuaris específics.
- Proporcionar suposicions negatives respecte a jugadors.
- Proporcionar canals alternatius per informar o avaluar a jugadors.
- Crear sistemes de lliga/*ranking* diferents dels oficials.
- Utilitzar mètodes per connectar-se a altres sistemes de *League of Legends*.
- Fer scrapping de qualsevol *endpoint* no documentat o d'altres fonts no proporcionades per *Riot API Endpoints*.
- Ser creat o dissenyat amb el mateix o similar disseny al natiu de *League of Legends* o qualsevol disseny o marca de *Riot Games*.

El projecte a desenvolupar ha de/pot:

- Tenir alguna manera per avaluar o millorar el joc dels mateixos jugadors.
- Connectar jugadors amb els seus amics.
- Utilitzar qualsevol dels recursos artístics del joc (però no els logos oficials).

5. Polítiques monetàries

L'API de *Riot Games* es proporciona com una eina per als desenvolupadors per crear un projecte que tots els jugadors pugui gaudir lliurement, no es tracta d'un mitjà bàsic per generar ingressos.

En cas de qualsevol dubte sobre la manera d'obtenir ingressos, s'ha de contactar directament amb els moderadors de *Riot Games*.

Anuncis: És el mètode preferit pels moderadors per obtenir ingressos pel projecte a realitzar. Si es rep el permís dels moderadors per tal de monetitzar el

projecte i es volen afegir anuncis, ha d'existir una opció per tal d'eliminar-los com a compra integrada a l'aplicació.

Per tal d'oferir l'eliminació de publicitat pagada, s'ha de:

- Contactar amb els moderadors mitjançant *App Note*.
- Proporcionar una visualització amb i sense anuncis. Acceptar els termes addicionals d'obtenció d'ingressos dins de l'aplicació del projecte.
- Accés exclusiu: No es pot cobrar diners per l'accés exclusiu a funcions que es basen, en la seva totalitat o en gran part, a les dades obtingudes de *l'API* de *Riot Games*.

6. Regles

Estan prohibides explícitament en la Comunitat de Desenvolupadors de *Riot Games* les següents accions:

- Comentaris ofensius relacionats amb el sexe, identitat i expressió de gènere, orientació sexual, discapacitat, malaltia mental, aparença física, tamany del cos, raça o religió.
- Comentaris no desitjats sobre les decisions i pràctiques d'estil de vida d'una persona.
- Contingut de *NSFW*.
- Amenaces de violència.
- Incitació a la violència cap a qualsevol individu, incloent-hi l'enfortiment d'una persona per suïcidar-se o per fer-se autolesions.
- Intimidació deliberada.
- Interrupció sostinguda del debat.
- Atenció sexual no desitjada.
- Continuació de la comunicació un-a-un després de les sol·licituds de cessament.
- Publicació de la comunicació privada sense consentiment.
- Es pretén ser un altre usuari, utilitzant el seu nom com a propi.
- Utilitzar noms ofensius/sobrenoms/icones.

7. Polítiques de dades

El projecte no pot allotjar cap sistema de classificació que pugui alterar o disminuir l'objectiu del joc. Es poden crear escales personalitzades o sistemes de rànquing sempre que no siguin reemplaçaments per als sistemes oficials. Si els sistemes d'un projecte alteren la jugabilitat o actuen com a substitut d'un sistema oficial, els moderadors poden demanar que el sistema o el projecte s'interrompi.

La destrucció del nexe sempre ha de ser l'objectiu primordial del joc. Un desafiament o una competència, l'objectiu del qual sigui qualsevol altra cosa que assolir una victòria està estrictament prohibit. Si un desafiament o un concurs personalitzat provoquen insatisfacció, frustració o enuig entre els companys d'equip, podem demanar que es discontinuï el repte, el concurs, el sistema o el projecte.

ANNEX D - DESCRIPCIÓ DE LES TECNOLOGIES UTILITZADES

1. *Geth*

Geth és una interfície per línia d'ordres el qual permet executar un node en alguna de les xarxes *Blockchain d'Ethereum*.

Aquest software permet participar en la xarxa *d'Ethereum* especificada en temps real. A més ens dóna l'opció de realitzar una gran quantitat de funcions. Les més representatives són:

- Crear i gestionar diferents comptes.
- Visualitzar la quantitat *d'Ethers* que té un compte.
- Crear contractes.
- Enviar transaccions.
- Transferir *Ethers* entre comptes.
- Minar *Ethers*.

En llançar una comanda per executar el node, *geth* sincronitza la *Blockchain d'Ethereum* especificada, i una vegada finalitzada tota la sincronització ja es podria executar qualsevol funció i/o realitzar qualsevol transacció.

A més, *Geth* ofereix múltiples interfícies: per línia d'ordres, un servidor *JSON-RPC* i una consola interactiva.

2. *Ganache*

Fig. D.1 Icona de *Ganache*

Ganache és una *Blockchain* personal i privada la qual està dissenyada per ajudar als programadors a desenvolupar qualsevol programa en la xarxa *Ethereum*.

Aquest software executa una *Blockchain* en local i ens crea diverses comptes amb una gran quantitat *d'Ethers*, la qual ens permet desplegar *Smart Contracts*, realitzar diferents test i llençar qualsevol transacció sense preocupar-nos del fet de quedar-se sense *Ethers*.

Ganache crea 10 comptes *d'Ethereum* amb 100 *Ethers* cada compte. A més, ens permet accedir no només a l'adreça de cada compte, sinó també a la seva clau privada.

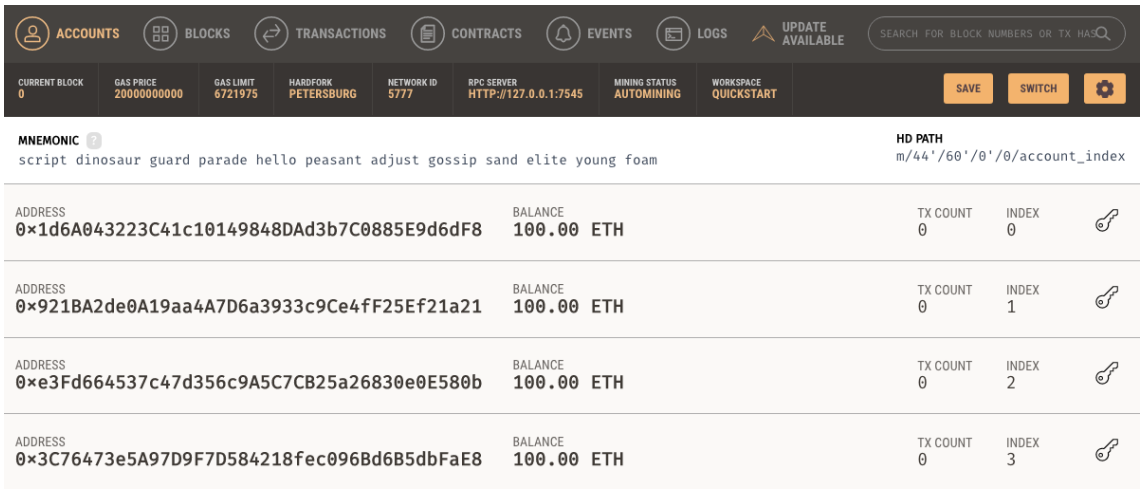


Fig. D.2 Adreces i balanços de cada compte

Una de les característiques principals d'aquest software és que no hi ha ningú que mini cap bloc, però, en canvi, qualsevol transacció realitzada correctament es confirma de manera immediata. Aquest fet permet realitzar qualsevol prova de manera senzilla i molt ràpida, sense la necessitat d'esperar el temps de confirmació d'una transacció que ens trobem si llancem una transacció a la xarxa principal *d'Ethereum* o en qualsevol *testnet*.

3. *MongoDB*

Fig. D.3 Icona de *MongoDB*

MongoDB és una base de dades *NOSQL* distribuïda de codi obert, que es basa en documents. Ofereix l'escalabilitat i la flexibilitat que es vulgui, a més de la possibilitat de realitzar qualsevol consulta que es necessiti i de forma senzilla.

L'estructura d'aquesta base de dades és la següent:

- Col·leccions: Cada base de dades està definida per diferents Col·leccions. Una col·lecció és un grup de documents els quals es volen organitzar dins d'aquesta col·lecció. Les col·leccions són anàlogues a les taules d'una base de dades relacional.
- Documents: Els documents són els registres de dades que es guarden en una col·lecció. Tenen una estructura de *BSON*, que es tracta d'una representació binària de documents *JSON*. El format *BSON*, com a diferència, conté més tipus de dades que el format *JSON*.
- Camps: Els camps són els elements que formen un document. Cada camp està definit per un tipus de dades, els quals poden ser: *Double, String, Object, Array, Binary data, Undefined, Object Id, Boolean, Date, Null, Regular Expression, Javascript, Symbol, Javascript with scope, Integer, Timestamp, Min key i Max key*.

Aquesta base de dades emmagatzema les dades introduïdes en forma de documents flexibles. Aquests documents tenen el format *JSON* i els camps de cada document no són fixes. Això ens permet que els camps dels documents variin un a un altre, i dóna l'opció de poder canviar l'estructura de dades en un futur.

El fet comentat anteriorment permet realitzar consultes a la base de dades de manera senzilla i poder accedir a les dades necessàries de forma ràpida i clara.

4. *Node JS*

Fig. D.4 Icona de *Node JS*

Node JS és un entorn de programació de codi obert, gratuït, amb capacitat de ser executat en diferents plataformes i que innova utilitzant *Javascript* en el servidor. Està dissenyat per escriure servidors web en *Javascript* i de manera asíncrona. A més, es basa en el llançament i rebuda *d'events*.

Node utilitza el motor *V8*, desenvolupat per *Google*, per executar *Javascript*. Aquest motor permet que *Node* tingui un entorn d'execució que compila i executa *Javascript* a altes velocitats en servidor.

Les principals funcions que proporciona *Node JS* són:

- Execució de codi *Javascript* en el *Back-End*, sense necessitat d'utilitzar un navegador.
- Crear, obrir, llegir, escriure, esborrar i tancar qualsevol arxiu en el servidor.
- Generar pàgines web de manera dinàmica.
- Gestionar, afegir, esborrar i modificar dades en una base de dades.

A més d'aquestes funcions, un dels apartats més forts que introdueix *Node JS* és el seu *Event Loop*. L'*Event Loop* és un únic procés el qual realitza totes les operacions d'entrada i sortida de manera asíncrona. Es tracta d'una cua de funcions que, en el moment que s'executa una funció, retorna el codi intern de la funció, ho envolta i ho introdueix a la cua.

5. *Express*

Fig. D.5 Icona de *Express JS*

Express és un *framework* de *Node JS*, de codi obert i escrit en *Javascript*, el qual s'utilitza principalment per a construir aplicacions web i *APIs*.

Els mecanismes que proporciona *Express* són:

- Realització i gestió de peticions *http*, anàlisi del cos de la sol·licitud, formació de la resposta i administració de les rutes.
- Creació i introducció de *Middlewares* en qualsevol punt de la petició.
- Gestió de *cookies*.
- Renderització de vistes mitjançant la introducció de dades.
- Ajustament de ports i localització de les respostes en renderitzar la resposta de la petició.

6. *Ionic*

Fig. D.6 Icona de *Ionic*

Ionic és un *framework* gratuït i de codi obert dissenyat per desenvolupar aplicacions multiplataforma per a *iOS*, Android i web amb només la necessitat de programar un únic codi base. *Ionic* és un paquet introduït dins de *NPM* Module i necessita la instal·lació prèvia de *Node JS*.

Ionic proporciona als desenvolupadors la possibilitat de programar aplicacions híbrides per móbil. Principalment, es basa en fer una barreja entre *Javascript/Typescript, HTML5 i CSS*. Seguidament, per poder realitzar la construcció de l'arxiu natiu de l'aplicació, *Ionic* dóna l'opció d'utilitzar *Apache Cordova* o *Capacitor*.

La primera versió de *Ionic* va ser construïda per sobre dels *frameworks AngularJS* i *Apache Cordova*. A continuació, la segona versió va avançar juntament amb el seu *framework* base, *Angular*, evolucionant en una programació basada en el llenguatge tipat *Typescript*.

Finalment, l'última versió de *Ionic, Ionic 4* permet utilitzar les *Web APIs* i està construït com un conjunt de *Web Components* que permet a l'usuari escollir el *framework* base en que programar, ja sigui *Angular, React* o *Vue JS*.

7. *Truffle*

Fig. D.7 Icona de *Truffle*

Truffle és un entorn de desenvolupament i una eina de proves que té com a objectiu principal donar una sèrie de mecanismes per ajudar a realitzar els projectes amb *Blockchain*.

Les funcions principals que ofereix *Truffle* són:

- Automatització dels desplegaments de *Smart Contracts*.
- Compilació de *Smart Contracts*
- Testejos de *Smart Contracts*.

8. *Mocha*

Fig. D.8 Icona de *Mocha*

Mocha és un *framework* de tests per *Node JS* encarregat de realitzar proves asíncrones de forma molt simple. A més, després de cada test elabora un informe precís detallant tant els errors com les proves realitzades correctament. Tots aquests tests, *Mocha* els permet executar tant en consola com des d'un navegador.

9. *Chai*

Fig. D.9 Icona de *Chai*

Chai és una biblioteca creada pel desenvolupament guiat per proves (*TDD*) i el desenvolupament guiat per comportament (*BDD*). L'objectiu dels mètodes *TDD* i *BDD* és assegurar la qualitat del programa a desenvolupar.

Chai s'executa tant per *Node JS* com per navegador, i permet realitzar tot tipus de proves incorporant-lo a qualsevol altre *framework* de *Javascript*. Paral·lelament, existeix *Chai http* que permet crear test realitzant peticions *http* per provar tots els mètodes necessaris.

10. *Apache Cordova*

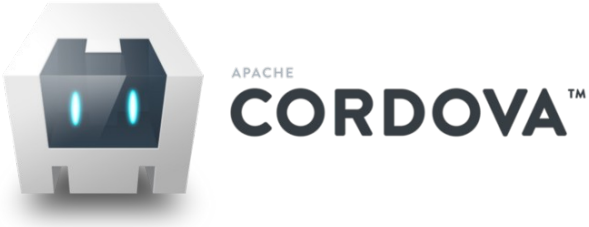


Fig. D.10 Icona de *Apache Cordova*

Apache Cordova és un *framework* de codi obert encarregat d'executar *HTML5, CSS i Javascript* per desenvolupar aplicacions en diverses plataformes. Permet transformar l'aplicació web mostrant-la dins d'una aplicació *WebView* nativa. Un WebView es tracta d'un component que s'utilitza per mostrar webs en contingut natiu.

L'objectiu principal d'*Apache Cordova* és construir aplicacions per a qualsevol plataforma, sigui Android, iOS o *Windows Phone*. Aquests tipus d'aplicacions que es desenvolupen en els llenguatges que podria executar qualsevol navegador però que s'interpreten en qualsevol plataforma reben el nom d'aplicacions híbrides.

ANNEX E - FLUXOS – FUNCIONALITATS FUTURES

En aquesta secció s'expliquen les funcionalitats futures per tal de crear una aposta i acceptar-la mitjançant l'ús de notificacions.

- L'usuari 1 crea l'aposta a partir d'una notificació:

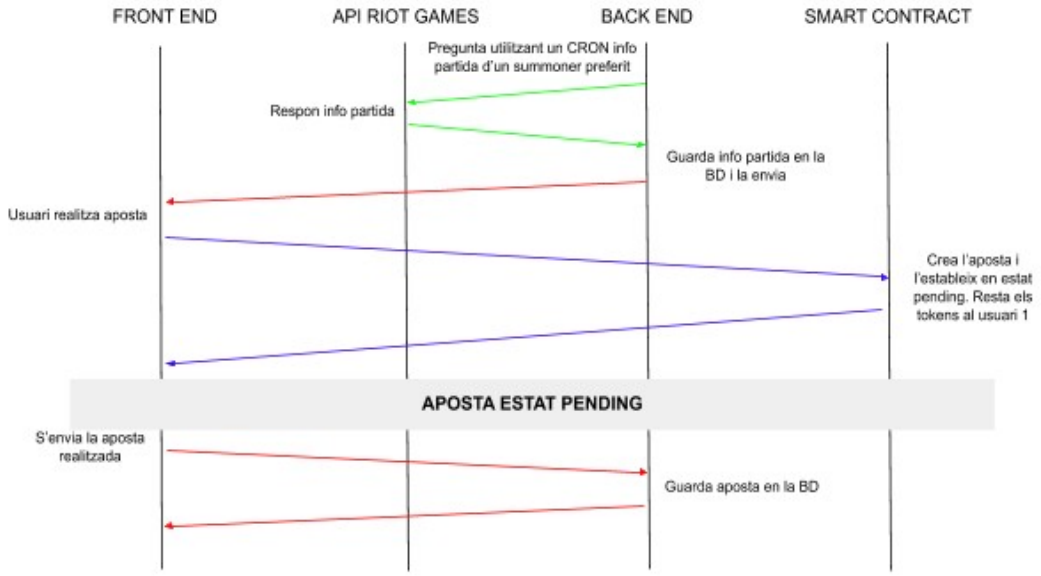


Fig. E.1 Diagrama de creació d'una aposta a partir d'una notificació

En aquesta situació, el *summoner* marcat com a preferit està jugant una partida.

En un futur, el *CRON* del *Back-End* preguntarà si el *summoner* preferit està jugant una partida, i en cas afirmatiu guarda la informació de la partida a la base de dades i envia una notificació cap a l'usuari.

A partir d'aquest punt, l'usuari 1 envia la transacció a la *Blockchain* executant el mètode *betCreate*() del *Smart Contract Bet.sol.*

El *Smart Contract* crea l'aposta i l'estableix en estat *Pending*. A més, actualitza el balanç de l'usuari 1 restant-li els *tokens* apostats.

Finalment, i una vegada s'ha confirmat la transacció de creació de l'aposta, es realitza una petició *http* cap a *l'API* del *Back-End* enviant l'aposta realitzada. El *Back-End* guarda l'aposta en la base de dades.

- L'usuari 2 rep una notificació informant que un amic ha creat una aposta o que algú ha creat una aposta d'una partida on està jugant el teu *summoner* preferit*:

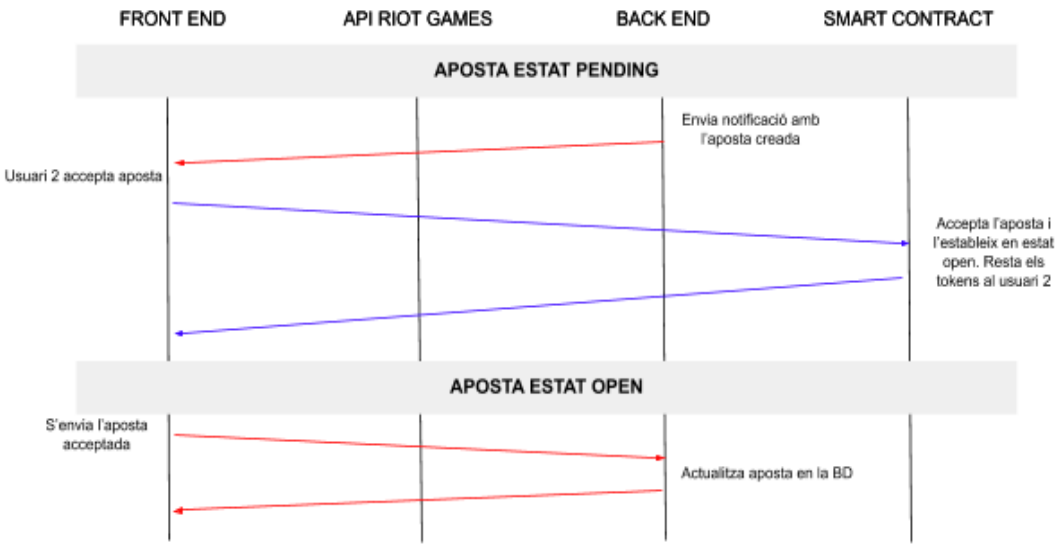


Fig. E.2 Diagrama d'acceptació a partir d'una notificació

En un futur, l'usuari 2 rep una notificació on és informat que un amic seu ha creat una aposta, o que qualsevol usuari ha creat una aposta on està jugant el seu *summoner* preferit.

L'usuari 2 en obrir la notificació pot visualitzar tota la informació de la partida i de l'aposta.

Posteriorment, l'usuari 2 accepta l'aposta i envia la transacció a la *Blockchain* executant el mètode *betOpen*() del *Smart Contract Bet.sol.*

El *Smart Contract* estableix l'aposta en l'estat Open. A més, actualitza el balanç de l'usuari 2 restant-li els *tokens* apostats.

ANNEX F - MODELS DE DADES A GUARDAR A MONGODB

En aquest annex es mostren tots els models de dades creats per emmagatzemar dades en la nostra base de dades.

- Bet:

```
let bet = mongoose.Schema({
        gameId: String,
        tokens: Number,
        duration: Number,
        summoner: String,
        state: String,
        id: Number,
        bettor1: String,
        bettor2: String,
        teamBettor1: String,
        teamBettor2: String,
        timestampBettor1: Number,
        timestampBettor2: Number,
        addressBettor1: String,
        addressBettor2: String,
        winner: String
    }, {
        collection: 'bet'
    }
);

module.exports = mongoose.model( name: 'Bet', bet);
```

Fig. F.1 Schema Bet de MongoDB

- Game;

```
let game = mongoose.Schema({
        gameId: String,
        gameStartTime: Number,
        gameMode: String,
        region: String,
        winner: String,
        participants: [{
            encryptedSummonerId: String,
            summonerName: String,
            team: Number, //100-blue 200-red
            championId: Number
        }]
    }, {
        collection: 'game'
    }
);

module.exports = mongoose.model( name: 'Game', game);
```

Fig. F.2 Schema Game de MongoDB

- Summoner;

```
let summoner = mongoose.Schema({
        summonerName: String,
        encryptedSummonerId: String,
        summonerLevel: Number
    }, {
        collection: 'summoner'
    }
);

module.exports = mongoose.model( name: 'Summoner', summoner);
```

Fig. F.3 Schema Summoner de MongoDB

- User;

```
let user = mongoose.Schema({
        username: String,
        password: String,
        email: String,
        address: String,
        token: String,
        reset_password_token: String,
        reset_password_expires: Date,
        friends: [],
        summoners: [],
        profilePhoto: String,
        stats: {
            total: Number,
            wins: Number,
            losses: Number,
            ratioWinLose: Number,
            userMostBets: String,
            youAreNemesisOf: String,
            yourNemesisIs: String
        },
        wallet: {}
    }, {
        collection: 'user'
    }
);

module.exports = mongoose.model( name: 'User', user);
```

Fig. F.4 Schema User de MongoDB

- Chat;

```
let chat = mongoose.Schema({
        room: String,
        users: [],
        created: Number,
        messages: [],
        lastMessage: String,
        lastMessageDate: Number
    }, {
        collection: 'chat'
    }
);

module.exports = mongoose.model( name: 'Chat', chat);
```

Fig. F.5 Schema Chat de MongoDB

- Message;

```
let message = mongoose.Schema({
        room: String,
        message: String,
        from: String,
        to: String,
        created: Number,
        seen: Boolean
    }, {
        collection: 'message'
    }
);

module.exports = mongoose.model( name: 'Message', message);
```

Fig. F.6 Schema Message de MongoDB

ANNEX G - FUNCIONS *SOCKET* – XAT

En aquest annex s'explica les funcions del xat on s'ha utilitzat els *sockets*.

- *Subscribe*:

```
socket.on('subscribe', async function(users) {
    let room;
    if(users && users.userFrom && users.userTo){
        if(users.userFrom.username && users.userTo.username) {
            if (users.userFrom.username.toLowerCase() < users.userTo.username.toLowerCase()) {
                room = "" + users.userFrom._id + "" + users.userTo._id;
            } else {
                room = "" + users.userTo._id + "" + users.userFrom._id;
            }
        }
        let checkChat = await Chat.findOne({ room: room });
        if(checkChat) {
            checkChat.users.find(user => {
                if(user.userId === users.userFrom._id) {
                    user.lastView = Date.now();
                }
            });
            await Chat.findOneAndUpdate({ room: room }, checkChat);

            console.log('joining room', room);
            socket.join(room);
        } else {
            let newChat = new Chat();
            newChat.room = room;
            newChat.created = Date.now();
            newChat.users.push({
                userId: users.userFrom._id,
                userName: users.userFrom.username,
                userConnected: users.userFrom.connected,
                lastView: Date.now()
            });
            newChat.users.push({
                userId: users.userTo._id,
                userName: users.userTo.username,
                userConnected: users.userTo.connected,
                lastView: null
            });
            await newChat.save();

            console.log('joining room', room);
            socket.join(room);
        }
    }
});
```

Fig. G.1 Funció subscribe() del socket

La funció *subscribe* s'executa en el moment que el *Front-End* emet *l'event subscribe*. En executar-se, el *Front-End* envia un *JSON* amb la informació necessària de cada usuari per tal de crear la sala de xat.

A continuació, es comprova si la sala ja està creada. En cas afirmatiu, es modifica l'hora d'última connexió i s'entra a la sala. I, en cas negatiu, es crea la sala, s'enregistra tota la informació dels usuaris en la base de dades i s'entra a la sala.

- *Disconnect*:

```
socket.on('disconnect', async function (username) {
    await User.findOneAndUpdate({name: username}, {connected: false}, {new: true});
    socket.emit('user-changed', {
        user: socket.username,
        event: 'left'
    })
});
```

Fig. G.2 Funció disconnect() del socket

La funció *disconnect* s'executa en el moment que el *Front-End* emet *l'event disconnect*. En executar-se, es modifica la base de dades informant que l'usuari ja no es troba connectat.

- *Set-username*:

```
socket.on('set-username', async (username) => {
    socket.username = username;
    await User.findOneAndUpdate({name: username}, {connected: true}, {new: true});
    socket.emit('users-changed', {
        user: username,
        event: 'joined'
    });
});
```

Fig. G.3 Funció set-username() del socket

La funció *set-username* s'executa en el moment que el *Front-End* emet *l'event set-username*. Aquesta funció s'executa en el moment que l'usuari es connecta, és en aquest punt en què es modifica la base de dades posant el camp *connected* a *true*.

- *Add-message*:

```
socket.on('add-message', async (message) => {
    console.log('message', message);
    let msg = new Message(message);
    let msgSaved = await msg.save();
    console.log('msgSaved', msgSaved);
    await Chat.updateOne({ room: message.room }, {
        $push: {
            messages: msgSaved._id
        },
    });
    await Chat.findOneAndUpdate({room: message.room}, {
        lastMessage: msgSaved.message,
        lastMessageDate: msgSaved.created
    });


    console.log('sending room post', message);
    socket.to(message.room).emit('message', msgSaved);
});
```

Fig. G.4 Funció add-message() del socket

La funció *add-message* s'executa en el moment que el *Front-End* emet *l'event add-message*. Aquesta funció s'executa en el moment que un dels dos usuaris envia un missatge. Es guarda el missatge i es modifica la informació relativa a l'últim missatge enviat a la sala.